ORBITER

Classification of Combinatorial Objects

Programmer's Guide
Build Number 3055

Anton Betten

January 30, 2023

Contents

Chapter 1

Examples

1.1 From the User's Guide

```
/*
 * ug_3_2_F_2.cpp
 *
 *  Created on: Jan 15, 2023
 *      Author: betten
 */




#include "orbiter.h"

using namespace std;
using namespace orbiter;


void first();
void second();
void third();

int main()
{
   orbiter::layer5_applications::user_interface::orbiter_top_level_session Orbiter
;

   first();
   second();
   third();
```

```
}

void first()
{
   field_theory::finite_field_description Descr;
   field_theory::finite_field Fq;

   int verbose_level = 2;

   Descr.f_q = TRUE;
   Descr.q_text.assign("2");
   Fq.init(&Descr, verbose_level);

   cout << "in F_2, 1 + 1 = " << Fq.add(1, 1) << endl;

   algebra::algebra_global Algebra;

   Algebra.do_cheat_sheet_GF(&Fq, verbose_level);
}

void second()
{
   int q = 2;
   int verbose_level = 2;
   int f_without_tables = FALSE;
   field_theory::finite_field Fq;

   Fq.finite_field_init_small_order(q,
         f_without_tables, verbose_level);

   cout << "in F_2, 1 + 1 = " << Fq.add(1, 1) << endl;

   algebra::algebra_global Algebra;

   Algebra.do_cheat_sheet_GF(&Fq, verbose_level);
}

void third()
{
   int q = 2;
   int verbose_level = 2;
   int f_without_tables = FALSE;
   field_theory::finite_field *Fq;

   Fq = NEW_OBJECT(field_theory::finite_field);

   Fq->finite_field_init_small_order(q,
```

```
        f_without_tables, verbose_level);

   cout << "in F_2, 1 + 1 = " << Fq->add(1, 1) << endl;


   algebra::algebra_global Algebra;

   Algebra.do_cheat_sheet_GF(Fq, verbose_level);

   FREE_OBJECT(Fq);
}
```

```
/*
 * ug_3_2_vandermonde.cpp
 *
 *  Created on: Jan 15, 2023
 *      Author: betten
 */

#include "orbiter.h"

using namespace std;
using namespace orbiter;



int main()
{

   orbiter::layer5_applications::user_interface::orbiter_top_level_session Orbiter
;

   int q = 7;
   int verbose_level = 2;
   int f_without_tables = FALSE;
   field_theory::finite_field Fq;

   Fq.finite_field_init_small_order(q,
         f_without_tables, verbose_level);

   int a;
   int i, j;
   int *V;
   int *W;

   V = NEW_int(q * q);
   W = NEW_int(q * q);

   for (i = 0; i < q; i++) {
      a = 1;
      V[i * q + 0] = 1;
      for (j = 1; j < q; j++) {
         a = Fq.mult(i, a);
         V[i * q + j] = a;
      }
   }

   cout << "Vandermonde matrix over F_" << q << endl;
   Int_matrix_print(V, q, q);
```

```
    Fq.Linear_algebra->invert_matrix(V, W, q, verbose_level);
    cout << endl;

    cout << "Inverse matrix:" << endl;
    Int_matrix_print(W, q, q);

    FREE_int(V);
    FREE_int(W);

}
```

```
/*
 * ug_5_3_quaternion.cpp
 *
 *  Created on: Jan 15, 2023
 *      Author: betten
 */



#include "orbiter.h"

using namespace std;
using namespace orbiter;



int main()
{

   orbiter::layer5_applications::user_interface::orbiter_top_level_session Orbiter
;

   int verbose_level = 2;
   int q = 3;
   int f_without_tables = FALSE;
   field_theory::finite_field *F;

   F = NEW_OBJECT(field_theory::finite_field);

   F->finite_field_init_small_order(q,
         f_without_tables, verbose_level);

   int gens[] = { 1,1,1,2, 2,1,1,1, 0,2,1,0 };


   actions::action *A;
   data_structures_groups::vector_ge *nice_gens;
   data_structures_groups::vector_ge *subgroup_gens;

   A = NEW_OBJECT(actions::action);

   A->init_general_linear_group(2, F,
         FALSE /*f_semilinear */, TRUE /* f_basis */, FALSE /* f_init_sims */,
         nice_gens,
         verbose_level);
```

```
groups::strong_generators *Gens;

Gens = NEW_OBJECT(groups::strong_generators);

ring_theory::longinteger_object target_go;

target_go.create(8, __FILE__, __LINE__);

Gens->init_from_data_with_target_go(A,
      gens,
      4, 3,
      target_go,
      subgroup_gens,
      verbose_level);

groups::sims *S;

S = Gens->create_sims(verbose_level);

long int go;
int i, j, k;
int *Elt1;
int *Elt2;
int *Elt3;

Elt1 = NEW_int(A->elt_size_in_int);
Elt2 = NEW_int(A->elt_size_in_int);
Elt3 = NEW_int(A->elt_size_in_int);
go = S->group_order_lint();
for (i = 0; i < go; i++) {
   S->element_unrank_lint(i, Elt1);
   for (j = 0; j < go; j++) {
      S->element_unrank_lint(j, Elt2);
      A->element_mult(Elt1, Elt2, Elt3, 0);
      k = S->element_rank_lint(Elt3);
      cout << k;
      if (j < go - 1) {
         cout << "\t";
      }
   }
   cout << endl;
}

FREE_int(Elt1);
FREE_int(Elt2);
FREE_int(Elt3);
FREE_OBJECT(Gens);
```

```
   FREE_OBJECT(S);
   FREE_OBJECT(A);
   FREE_OBJECT(F);

}
```

Chapter 2

The Library

2.1 Main Header File

```
// orbiter.h
//
// Anton Betten
//
// started:  October 23, 2002
// 2nd version started:  December 7, 2003
// INCIDENCE included: August 25, 2007
// SNAKES_AND_LADDERS started: September 20, 2007
// TOP_LEVEL started: September 23, 2010
// INCIDENCE deleted: July 27, 2018

#ifndef _ORBITER_
#define _ORBITER_


/*! \mainpage Orbiter - Classification of Combinatorial Objects
 *
 * \section users_guide_sec User´s Guide
 *
 * <A HREF="http://www.math.colostate.edu/~betten/orbiter/users_guide.pdf"> Orbit
er User´s Guide </A>
 *
 * \section programmers_guide_sec Programmer´s Guide
 *
 * <A HREF="https://www.math.colostate.edu/~betten/orbiter/orbiter_programmers_gu
ide.pdf"> Orbiter Programmer´s Guide </A>
 *
 * \section github_sec Sources on GitHub
 *
 * <A HREF="https://github.com/abetten/orbiter"> Orbiter Sources on GitHub </A>
 *
```

```
 * \section team_sec Orbiter Team members
 *
 * Anton Betten, Abdullah AlAzemi, Fatma Karaoglu, Sajeeb Chowdhury. Many contrib
utors not listed here.
 *
 *
 *
 */


#include "layer1_foundations/foundations.h"
#include "layer2_discreta/discreta.h"
#include "layer3_group_actions/group_actions.h"
#include "layer4_classification/classification.h"
#include "layer5_top_level/top_level.h"

#endif
```

Chapter 3

Layer 1 – Foundations

3.1 Main Header File

```
// foundations.h
//
// Anton Betten
//
// renamed from galois.h: August 16, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005


#ifndef ORBITER_SRC_LIB_FOUNDATIONS_FOUNDATIONS_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_FOUNDATIONS_H_



// History:
//
// added class unipoly_domain: November 16, 2002
// added class finite_field: October 23, 2002
// added class longinteger: October 26, 2002
// added class mp: March 6, 2003
// added partitionstack: July 3 2007
// added class orthogonal: July 9 2007
// added class vector_hashing: October 14, 2008
// added file tensor: Dec 25 2008
// added class grassmann: June 5, 2009
// added class unusual: June 10 2009
// added file memory: June 25, 2009
// added file geometry: July 9, 2009
// added class classify: Oct 31, 2009
// added class grassmann_embedded: Jan 24, 2010
```

```
// added class hermitian: March 19, 2010
// added class incidence_structure: June 20, 2010
// added class finite_ring: June 21, 2010
// added class hjelmslev: June 22, 2010
// added class fancy_set: June 29, 2010
// added class norm_tables: Sept 23, 2010 (started 11/28/2008)
// added struct grid_frame: Sept 8, 2011
// added class data_file: Oct 13, 2011
// added class subfield_structure: November 14, 2011
// added class clique_finder: December 13, 2011
// added class colored_graph: October 28, 2012
// added class rainbow_cliques: October 28, 2012
// added class set_of_sets: November 30, 2012
// added class decomposition: December 1, 2012
// added file dlx.cpp: April 7, 2013
// added class spreadsheet: March 15, 2013
// added class andre_construction andre_construction: June 2, 2013
// added class andre_construction_point_element: June 2, 2013
// added class andre_construction_line_element: June 2, 2013
// added class int_matrix: October 23, 2013
// added class gl_classes: October 23, 2013
// added class layered_graph: January 6, 2014
// added class graph_layer: January 6, 2014
// added class graph_node: January 6, 2014
// added class int_vector: August 12, 2014
// added class projective_space (moved here from ACTION): December 31, 2014
// added class buekenhout_metz (moved here from TOP_LEVEL): December 31, 2014
// added class a_domain March 14, 2015
// added class diophant (moved here from INCIDENCE) April 16, 2015
// added class null_polarity_generator December 11, 2015
// added class layered_graph_draw_options December 15, 2015
// added class klein_correspondence January 1, 2016
// added class file_output January 8, 2016
// added class generators_symplectic_group March 29, 2016
// added class flag May 20, 2016
// moved class knarr from TOP_LEVEL: Jul 29, 2016
// moved class w3q from TOP_LEVEL: Jul 29, 2016
// moved class surface from TOP_LEVEL: Aug 1, 2016
// added class homogeneous_polynomial_domain: Sept 9, 2016
// added class eckardt_point: January 12, 2017
// added class surface_object: March 18, 2017
// added class vector_space: December 2, 2018

#include <iostream>
#include <fstream>
#include <iomanip>
```

```
#include <cstring>
#include <math.h>
#include <limits.h>
#include <sstream>

#include <map>
#include <vector>
#include <deque>
#include <string>




/*-------------------------------------------------------------------*/
/// Define some ANSI colour codes
/*-------------------------------------------------------------------*/
#if __cplusplus >= 201103L
#define COLOR_UNICODE "\u001b"
#else
#define COLOR_UNICODE "\x1b"
#endif

#define RESET_COLOR_SCHEME COLOR_UNICODE "[0m"
#define BLACK COLOR_UNICODE "[30m"
#define RED COLOR_UNICODE "[31m"
#define GREEN COLOR_UNICODE "[32m"
#define YELLOW COLOR_UNICODE "[33m"
#define BLUE COLOR_UNICODE "[34m"
#define MAGENTA COLOR_UNICODE "[35m"
#define CYAN COLOR_UNICODE "[36m"
#define WHITE COLOR_UNICODE "[37m"

#define BRIGHT_BLACK COLOR_UNICODE "[30;1m"
#define BRIGHT_RED COLOR_UNICODE "[31;1m"
#define BRIGHT_GREEN COLOR_UNICODE "[32;1m"
#define BRIGHT_YELLOW COLOR_UNICODE "[33;1m"
#define BRIGHT_BLUE COLOR_UNICODE "[34;1m"
#define BRIGHT_MAGENTA COLOR_UNICODE "[35;1m"
#define BRIGHT_CYAN COLOR_UNICODE "[36;1m"
#define BRIGHT_WHITE COLOR_UNICODE "[37;1m"
/*-------------------------------------------------------------------*/


/*-------------------------------------------------------------------*/
/// The following code block identifies the current operating system the code is
/// being executed on and turns on specific macros in order to use system calls
```

```
/// defined by that operating system.
/*-----------------------------------------------------------------*/
#if defined(unix) || defined(__unix) || defined(__unix__)
#define SYSTEMUNIX
#endif

#if defined(_WIN32) || defined(_WIN64)
#define SYSTEMWINDOWS
#endif

#if defined(__APPLE__) || defined(__MACH__)
#define SYSTEMUNIX
#define SYSTEM_IS_MACINTOSH
   // use Mac specific stuff like asking how much memory the process uses.
#endif

#if defined(__linux__) || defined(linux) || defined(__linux)
#define SYSTEM_LINUX
#endif
/*-----------------------------------------------------------------*/

#define SYSTEMUNIX


#define HAS_NAUTY 1

// need to be defined in nauty_interface.cpp also.


#ifdef SYSTEMWINDOWS
//#pragma warning(disable : 4996)
#include <string>
#endif

#define MEMORY_DEBUG

#define MAGIC_SYNC 762873656L



// define exactly one of the following to match your system:
#undef int_HAS_2_charS
#define int_HAS_4_charS
#undef int_HAS_8_charS

#ifdef int_HAS_2_charS
typedef short int_2;
```

```
typedef long int_4;
typedef long int_8;
typedef unsigned short uint_2;
typedef unsigned long uint_4;
typedef unsigned long uint_8;
#endif
#ifdef int_HAS_4_charS
typedef short int_2;
typedef int int_4;
typedef long int_8;
typedef unsigned short uint_2;
typedef unsigned int uint_4;
typedef unsigned long uint_8;
#endif
#ifdef int_HAS_8_charS
typedef short int_2;
typedef short int int_4;
typedef int int_8;
typedef unsigned short uint_2;
typedef unsigned short int uint_4;
typedef unsigned int uint_8;
#endif


typedef int *pint;
typedef long int *plint;
typedef int **ppint;
typedef long int **pplint;
typedef char *pchar;
typedef unsigned char uchar;
typedef uchar *puchar;
typedef void *pvoid;



#define PAGE_LENGTH_LOG 20
#define PAGE_STORAGE_MAX_PAGE_SIZE (5 * 1L << 20)
#define BUFSIZE 100000


#define MINIMUM(x, y)   ( ((x) < (y)) ?  (x) : (y) )
#define MAXIMUM(x, y)   ( ((x) > (y)) ?  (x) : (y) )
#define MIN(x, y)   ( ((x) < (y)) ?  (x) : (y) )
#define MAX(x, y)   ( ((x) > (y)) ?  (x) : (y) )
#define ABS(x)      ( ((x) <  0 ) ? (-(x)) : (x) )
#define EVEN(x)     ( ((x) % 2) == 0 )
#define ODD(x)      ( ((x) % 2) == 1 )
```

```
#define DOUBLYEVEN(x)     ( ((x) % 4) == 0 )
#define SINGLYEVEN(x)     ( ((x) % 4) == 2 )
#define ONE_MILLION 1000000
#define ONE_HUNDRED_THOUSAND 100000


#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif
#ifndef M_PI
#define M_PI 3.14159265358979323846264
#endif


//! the orbiter library for the classification of combinatorial objects

namespace orbiter {



//! algebra, combinatorics and graph theory, geometry, linear algebra, number the
ory, data structures, solvers, graphics; no group actions

namespace layer1_foundations {



//! algebraic algorithms, generators for certain classes of groups, conjugacy cla
sses in the general linear group

namespace algebra {

   // algebra:
   class a_domain;
   class algebra_global;
   class generators_symplectic_group;
   class gl_class_rep;
   class gl_classes;
   class group_generators_domain;
   class heisenberg;
   class interface_gap_low;
   class interface_magma_low;
   class matrix_block_data;
   class null_polarity_generator;
```

```
   class rank_checker;
   class vector_space;

}


//! cubic surfaces, quartic curves, Schlaefli labelings, Eckardt points,  Del Pez
zo surfaces, Clebsch maps

namespace algebraic_geometry {

   class algebraic_geometry_global;
   class arc_lifting_with_two_lines;
   class clebsch_map;
   class cubic_curve;
   class del_pezzo_surface_of_degree_two_domain;
   class del_pezzo_surface_of_degree_two_object;
   class eckardt_point_info;
   class eckardt_point;
   class quartic_curve_domain;
   class quartic_curve_object_properties;
   class quartic_curve_object;
   class schlaefli_labels;
   class schlaefli;
   class seventytwo_cases;
   class smooth_surface_object_properties;
   class surface_domain;
   class surface_object_properties;
   class surface_object;
   class surface_polynomial_domains;
   class web_of_cubic_curves;

}

//! coding theory, MacWilliams, weight enumerators, cyclic codes, BCH codes, Reed
-Muller codes, etc.

namespace coding_theory {

   // coding_theory:
   class code_diagram;
   class coding_theory_domain;
   class crc_codes;
   class crc_options_description;
   class create_BCH_code;
   class create_RS_code;
   class error_repository;
```

```
   class ttp_codes;

}

//! combinatorics: boolean functions, combinatorial objects, classification, tact
ical decompositions, various puzzles

namespace combinatorics {

   // combinatorics:
   class boolean_function_domain;
   class brick_domain;
   class classification_of_objects_description;
   class classification_of_objects_report_options;
   class classification_of_objects;
   class combinatorics_domain;
   class domino_assignment;
   class domino_change;
   class encoded_combinatorial_object;
   class geo_parameter;
   class pentomino_puzzle;
   class polynomial_function_domain;
   class tdo_data;
   class tdo_refinement_description;
   class tdo_refinement;
   class tdo_scheme_compute;
   class tdo_scheme_synthetic;
   struct solution_file_data;

}


//! cryptography: Vigenere, Caesar, RSA, primality tests, elliptic curve, NTRU, s
quare roots modulo n.

namespace cryptography {

   // cryptography
   class cryptography_domain;

}

//! basic data structures used throughout the project

namespace data_structures {

   // data_structures:
```

```
   class algorithms;
   class bitmatrix;
   class bitvector;
   class classify_bitvectors;
   class classify_using_canonical_forms;
   class data_file;
   class data_input_stream_description;
   class data_input_stream;
   class data_structures_global;
   class fancy_set;
   class int_matrix;
   class int_vec;
   class int_vector;
   class lint_vec;
   class nauty_output;
   class page_storage;
   class partitionstack;
   class set_builder_description;
   class set_builder;
   class set_of_sets_lint;
   class set_of_sets;
   class sorting;
   class spreadsheet;
   class string_tools;
   class tally_lint;
   class tally_vector_data;
   class tally;
   class vector_hashing;

}


//! expression parser, used to create an abstract syntax tree (AST) of a well-for
med algebraic expression

namespace expression_parser {

   // expression_parser:
   class expression_parser_domain;
   class expression_parser;
   class formula_activity_description;
   class formula_activity;
   class formula;
   class lexer;
   class syntax_tree_node_terminal;
   class syntax_tree_node;
   class syntax_tree;
```

```
}


//! finite fields, n-th roots, subfields, trace and norm.

namespace field_theory {

   // finite_fields:
   class finite_field_activity_description;
   class finite_field_activity;
   class finite_field_description;
   class finite_field;
   class finite_field_implementation_by_tables;
   class finite_field_implementation_wo_tables;
   class minimum_polynomial;
   class norm_tables;
   class nth_roots;
   class related_fields;
   class square_nonsquare;
   class subfield_structure;
}


//! projective geometry over a finite field and related topics

namespace geometry {


   // geometry:
   class andre_construction_line_element;
   class andre_construction_point_element;
   class andre_construction;
   class arc_basic;
   class arc_in_projective_space;
   class buekenhout_metz;
   class decomposition;
   class desarguesian_spread;
   class flag;
   class geometric_object_create;
   class geometric_object_description;
   class geometry_global;
   class grassmann_embedded;
   class grassmann;
   class hermitian;
   class hjelmslev;
   class incidence_structure;
```

```
   class intersection_type;
   class klein_correspondence;
   class knarr;
   class object_with_canonical_form;
   class point_line;
   class points_and_lines;
   class polarity;
   class projective_space_implementation;
   class projective_space_of_dimension_three;
   class projective_space_plane;
   class projective_space_reporting;
   class projective_space;
   class spread_domain;
   class spread_tables;
   class W3q;

}

//! construction and classification of configurations, linear spaces, and designs


namespace geometry_builder {

   // geometry_builder:
   class cperm;
   class decomposition_with_fuse;
   class gen_geo_conf;
   class gen_geo;
   class geometric_backtrack_search;
   class geometry_builder_description;
   class geometry_builder;
   class girth_test;
   class inc_encoding;
   class incidence;
   class iso_type;
   class test_semicanonical;

}



//! graph theory: constructions, clique finding, drawing

namespace graph_theory {

   // graph_theory
   class clique_finder_control;
```

```
   class clique_finder;
   class colored_graph;
   class graph_layer;
   class graph_node;
   class graph_theory_domain;
   class layered_graph;
   class rainbow_cliques;

}

// graph_theory_nauty
class nauty_interface;

//! graphical output interfaces: 2D graphics (BMP, TikZ, Metapost) and 3D graphic
s (povray)

namespace graphics {

   // graphics:
   class animate;
   class draw_bitmap_control;
   class draw_incidence_structure_description;
   class draw_mod_n_description;
   class draw_projective_curve_description;
   class drawable_set_of_objects;
   class graphical_output;
   class layered_graph_draw_options;
   class mp_graphics;
   class parametric_curve_point;
   class parametric_curve;
   class plot_tools;
   class povray_interface;
   class povray_job_description;
   class scene_element_of_type_edge;
   class scene_element_of_type_face;
   class scene_element_of_type_line;
   class scene_element_of_type_plane;
   class scene_element_of_type_point;
   class scene_element_of_type_surface;
   class scene;
   class tree_draw_options;
   class tree;
   class tree_node;
   class video_draw_options;
   // pointer types
   typedef tree_node *ptree_node;
```

```
}


//! database of mathematical objects

namespace knowledge_base {

   // knowledge_base:
   class knowledge_base;

}


//! linear algebra and representation theory

namespace linear_algebra {

   // linear_algebra:
   class linear_algebra;
   class representation_theory_domain;

}

//! number theory, cyclotomic sets, elliptic curves, number theoretic transform (
NTT)

namespace number_theory {

   // number_theory:
   class cyclotomic_sets;
   class elliptic_curve;
   class number_theoretic_transform;
   class number_theory_domain;

}

//! the Orbiter kernel. It contains functions related to the symbol-table, memory
 management, os-interface, file-io, latex-interface etc.

namespace orbiter_kernel_system {

   class create_file_description;
   class file_io;
   class file_output;
   class latex_interface;
   class mem_object_registry_entry;
   class mem_object_registry;
```

```
   class memory_object;
   class numerics;
   class orbiter_data_file;
   class orbiter_session;
   class orbiter_symbol_table_entry;
   class orbiter_symbol_table;
   class os_interface;
   class override_double;
   class prepare_frames;

}


//! orthogonal geometry: quadrics, BLT sets

namespace orthogonal_geometry {

   // orthogonal:
   class blt_set_domain;
   class blt_set_invariants;
   class linear_complex;
   class orthogonal_global;
   class orthogonal_group;
   class orthogonal_indexing;
   class orthogonal;
   class quadratic_form_list_coding;
   class quadratic_form;
   class unusual_model;

}


//! expressions in reverse Polish notation

namespace polish {

   class function_command;
   class function_polish_description;
   class function_polish;

}


//! ring theory, including polynomial rings and longinteger arithmetic.

namespace ring_theory {
   // ring_theory:
```

```
   class finite_ring;
   class homogeneous_polynomial_domain;
   class longinteger_domain;
   class longinteger_object;
   class partial_derivative;
   class polynomial_double_domain;
   class polynomial_double;
   class polynomial_ring_activity_description;
   class polynomial_ring_description;
   class ring_theory_global;
   class table_of_irreducible_polynomials;
   class unipoly_domain;

   typedef ring_theory::longinteger_object *plonginteger_object;
   typedef void *unipoly_object;

}

//! diophantine systems of equations. Solvers Possolve and Dancing Links.

namespace solvers {
   // solvers
   class diophant_activity_description;
   class diophant_activity;
   class diophant_create;
   class diophant_description;
   class diophant;
   class dlx_problem_description;
   class dlx_solver;
   struct dlx_node;
   typedef struct dlx_node *pdlx_node;
}




#ifdef MEMORY_DEBUG
#define NEW_int(n) orbiter_kernel_system::Orbiter->global_mem_object_registry->al
locate_int(n, __FILE__, __LINE__)
#define NEW_int_with_tracking(n, file, line) orbiter_kernel_system::Orbiter->glob
al_mem_object_registry->allocate_int(n, file, line)
#define NEW_pint(n) orbiter_kernel_system::Orbiter->global_mem_object_registry->a
llocate_pint(n, __FILE__, __LINE__)
#define NEW_lint(n) orbiter_kernel_system::Orbiter->global_mem_object_registry->a
llocate_lint(n, __FILE__, __LINE__)
#define NEW_plint(n) orbiter_kernel_system::Orbiter->global_mem_object_registry->
```

```
allocate_plint(n, __FILE__, __LINE__)
#define NEW_ppint(n) orbiter_kernel_system::Orbiter->global_mem_object_registry->
allocate_ppint(n, __FILE__, __LINE__)
#define NEW_pplint(n) orbiter_kernel_system::Orbiter->global_mem_object_registry-
>allocate_pplint(n, __FILE__, __LINE__)
#define NEW_char(n) orbiter_kernel_system::Orbiter->global_mem_object_registry->a
llocate_char(n, __FILE__, __LINE__)
#define NEW_char_with_tracking(n, file, line) orbiter_kernel_system::Orbiter->glo
bal_mem_object_registry->allocate_char(n, file, line)
#define NEW_uchar(n) orbiter_kernel_system::Orbiter->global_mem_object_registry->
allocate_uchar(n, __FILE__, __LINE__)
#define NEW_pchar(n) orbiter_kernel_system::Orbiter->global_mem_object_registry->
allocate_pchar(n, __FILE__, __LINE__)
#define NEW_puchar(n) orbiter_kernel_system::Orbiter->global_mem_object_registry-
>allocate_puchar(n, __FILE__, __LINE__)
#define NEW_pvoid(n) orbiter_kernel_system::Orbiter->global_mem_object_registry->
allocate_pvoid(n, __FILE__, __LINE__)
#define NEW_OBJECT(type) (type *)orbiter_kernel_system::Orbiter->global_mem_objec
t_registry->allocate_OBJECT(new type, (std::size_t) sizeof(type), #type, __FILE__
, __LINE__)
#define NEW_OBJECTS(type, n) (type *)orbiter_kernel_system::Orbiter->global_mem_o
bject_registry->allocate_OBJECTS(new type[n], n, (std::size_t) sizeof(type), #typ
e, __FILE__, __LINE__)
#define FREE_int(p) orbiter_kernel_system::Orbiter->global_mem_object_registry->f
ree_int(p, __FILE__, __LINE__)
#define FREE_pint(p) orbiter_kernel_system::Orbiter->global_mem_object_registry->
free_pint(p, __FILE__, __LINE__)
#define FREE_lint(p) orbiter_kernel_system::Orbiter->global_mem_object_registry->
free_lint(p, __FILE__, __LINE__)
#define FREE_plint(p) orbiter_kernel_system::Orbiter->global_mem_object_registry-
>free_plint(p, __FILE__, __LINE__)
#define FREE_ppint(p) orbiter_kernel_system::Orbiter->global_mem_object_registry-
>free_ppint(p, __FILE__, __LINE__)
#define FREE_pplint(p) orbiter_kernel_system::Orbiter->global_mem_object_registry
->free_pplint(p, __FILE__, __LINE__)
#define FREE_char(p) orbiter_kernel_system::Orbiter->global_mem_object_registry->
free_char(p, __FILE__, __LINE__)
#define FREE_uchar(p) orbiter_kernel_system::Orbiter->global_mem_object_registry-
>free_uchar(p, __FILE__, __LINE__)
#define FREE_pchar(p) orbiter_kernel_system::Orbiter->global_mem_object_registry-
>free_pchar(p, __FILE__, __LINE__)
#define FREE_puchar(p) orbiter_kernel_system::Orbiter->global_mem_object_registry
->free_puchar(p, __FILE__, __LINE__)
#define FREE_pvoid(p) orbiter_kernel_system::Orbiter->global_mem_object_registry-
>free_pvoid(p, __FILE__, __LINE__)
#define FREE_OBJECT(p) {orbiter_kernel_system::Orbiter->global_mem_object_registr
y->free_OBJECT(p, __FILE__, __LINE__); delete p;}
```

```
#define FREE_OBJECTS(p) {orbiter_kernel_system::Orbiter->global_mem_object_regist
ry->free_OBJECTS(p, __FILE__, __LINE__); delete [] p;}
#else
#define NEW_int(n) new int[n]
#define NEW_int_with_tracking(n, file, line) new int[n]
#define NEW_pint(n) new pint[n]
#define NEW_lint(n) new long int[n]
#define NEW_lint(n) new (long int *)[n]
#define NEW_ppint(n) new ppint[n]
#define NEW_pplint(n) new pplint[n]
#define NEW_char(n) new char[n]
#define NEW_char_with_tracking(n, file, line) new char[n]
#define NEW_uchar(n) new uchar[n]
#define NEW_pchar(n) new pchar[n]
#define NEW_puchar(n) new puchar[n]
#define NEW_pvoid(n) new pvoid[n]
#define NEW_OBJECT(type) new type
#define NEW_OBJECTS(type, n) new type[n]
#define FREE_int(p) delete [] p
#define FREE_pint(p) delete [] p
#define FREE_lint(p) delete [] p
#define FREE_plint(p) delete [] p
#define FREE_ppint(p) delete [] p
#define FREE_pplint(p) delete [] p
#define FREE_char(p) delete [] p
#define FREE_uchar(p) delete [] p
#define FREE_pchar(p) delete [] p
#define FREE_puchar(p) delete [] p
#define FREE_pvoid(p) delete [] p
#define FREE_OBJECT(p) delete p
#define FREE_OBJECTS(p) delete [] p
#endif


#define Int_vec_print(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->print(A,
B, C)
#define Lint_vec_print(A, B, C) orbiter_kernel_system::Orbiter->Lint_vec->print(A
, B, C)
#define Int_vec_print_fully(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->pri
nt_fully(A, B, C)
#define Lint_vec_print_fully(A, B, C) orbiter_kernel_system::Orbiter->Lint_vec->p
rint_fully(A, B, C)

#define Int_vec_print_integer_matrix(A,B,C,D) orbiter_kernel_system::Orbiter->Int
_vec->print_integer_matrix(A, B, C, D)
#define Int_vec_print_integer_matrix_width(A,B,C,D,E,F) orbiter_kernel_system::Or
biter->Int_vec->print_integer_matrix_width(A, B, C, D, E, F)
```

```
#define Int_vec_copy(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->copy(A, B,
 C)
#define Lint_vec_copy(A, B, C) orbiter_kernel_system::Orbiter->Lint_vec->copy(A,
B, C)

#define Int_vec_print_to_str(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->pr
int_to_str(A, B, C)
#define Lint_vec_print_to_str(A, B, C) orbiter_kernel_system::Orbiter->Lint_vec->
print_to_str(A, B, C)


#define Int_vec_print_str_naked(A, B, C) orbiter_kernel_system::Orbiter->Int_vec-
>print_str_naked(A, B, C)


#define Int_vec_print_GAP(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->print
_GAP(A, B, C)
#define Lint_vec_print_GAP(A, B, C) orbiter_kernel_system::Orbiter->Lint_vec->pri
nt_GAP(A, B, C)


#define Int_matrix_print(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->matrix
_print(A, B, C)
#define Lint_matrix_print(A, B, C) orbiter_kernel_system::Orbiter->Lint_vec->matr
ix_print(A, B, C)

#define Int_matrix_print_ost(A, B, C, D) orbiter_kernel_system::Orbiter->Int_vec-
>matrix_print_ost(A, B, C, D)

#define Int_matrix_print_bitwise(A, B, C) orbiter_kernel_system::Orbiter->Int_vec
->matrix_print_bitwise(A, B, C)


#define Int_vec_zero(A, B) orbiter_kernel_system::Orbiter->Int_vec->zero(A, B);
#define Lint_vec_zero(A, B) orbiter_kernel_system::Orbiter->Lint_vec->zero(A, B)

#define Int_vec_is_zero(A, B) orbiter_kernel_system::Orbiter->Int_vec->is_zero(A,
 B)

#define Int_vec_scan(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->scan(A, B,
 C)
#define Lint_vec_scan(A, B, C) orbiter_kernel_system::Orbiter->Lint_vec->scan(A,
B, C)


#define Int_vec_copy_to_lint(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->co
```

```
py_to_lint(A, B, C)
#define Lint_vec_copy_to_int(A, B, C) orbiter_kernel_system::Orbiter->Lint_vec->c
opy_to_int(A, B, C)


#define Int_vec_print_integer_matrix_in_C_source(A, B, C, D) orbiter_kernel_syste
m::Orbiter->Int_vec->print_integer_matrix_in_C_source(A, B, C, D)

#define Int_vec_apply_lint(A, B, C, D) orbiter_kernel_system::Orbiter->Int_vec->a
pply_lint(A, B, C, D)

#define Int_vec_mone(A, B) orbiter_kernel_system::Orbiter->Int_vec->mone(A, B)

#define Int_vec_print_Cpp(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->print
_Cpp(A, B, C)

#define Int_vec_complement(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->comp
lement(A, B, C)
#define Int_vec_complement_to(A, B, C, D) orbiter_kernel_system::Orbiter->Int_vec
->complement(A, B, C, D)

#define Lint_vec_complement_to(A, B, C, D) orbiter_kernel_system::Orbiter->Lint_v
ec->complement(A, B, C, D)


#define Int_vec_set_print(A, B, C) orbiter_kernel_system::Orbiter->Int_vec->set_p
rint(A, B, C)



#define Int_vec_print_classified_str(A, B, C, D) orbiter_kernel_system::Orbiter->
Int_vec->print_classified_str(A, B, C, D)

#define Int_vec_distribution(A, B, C, D, E) orbiter_kernel_system::Orbiter->Int_v
ec->distribution(A, B, C, D, E)

#define Int_vec_find_first_nonzero_entry(A, B) pivot = orbiter_kernel_system::Orb
iter->Int_vec->find_first_nonzero_entry(A, B)


#define Int_vec_create_string_with_quotes(str, v, len) orbiter_kernel_system::Orb
iter->Int_vec->create_string_with_quotes(str, v, len)
#define Lint_vec_create_string_with_quotes(str, v, len) orbiter_kernel_system::Or
biter->Lint_vec->create_string_with_quotes(str, v, len)


#define Get_int_vector_from_label(A, B, C, D) orbiter_kernel_system::Orbiter->get
```

```
_int_vector_from_label(A, B, C, D)
#define Get_lint_vector_from_label(A, B, C, D) orbiter_kernel_system::Orbiter->ge
t_lint_vector_from_label(A, B, C, D)
#define Get_matrix(label, A, m, n) orbiter_kernel_system::Orbiter->get_matrix_fro
m_label(label, A, m, n)
#define Get_ring(label) orbiter_kernel_system::Orbiter->get_object_of_type_polyno
mial_ring(label)
#define Get_finite_field(label) orbiter_kernel_system::Orbiter->get_object_of_typ
e_finite_field(label)




enum monomial_ordering_type {
   t_LEX, // lexicographical
   t_PART, // by partition type
};


enum object_with_canonical_form_type {
   t_PTS, // points
   t_LNS, // lines
   t_PNL, // points and lines
   t_PAC, // packing
   t_INC, // incidence geometry
   t_LS // large set
};

enum diophant_equation_type {
   t_EQ, // equal to the given value
   t_LE, // less than or equal to the given value
   t_INT, // must be within the given interval
   t_ZOR // zero or equal to the given value
};

enum symbol_table_object_type {
   t_nothing_object,
   t_finite_field,
   t_polynomial_ring,
   t_any_group,
   t_linear_group,
   t_permutation_group,
   t_modified_group,
   t_projective_space,
   t_orthogonal_space,
   t_BLT_set_classify,
   t_spread_classify,
```

```
   t_formula,
   t_cubic_surface,
   t_quartic_curve,
   t_BLT_set,
   t_classification_of_cubic_surfaces_with_double_sixes,
   t_collection,
   t_geometric_object,
   t_graph,
   t_code,
   t_spread,
   t_translation_plane,
   t_spread_table,
   t_packing_was,
   t_packing_was_choose_fixed_points,
   t_packing_long_orbits,
   t_graph_classify,
   t_diophant,
   t_design,
   t_design_table,
   t_large_set_was,
   t_set,
   t_vector,
   t_combinatorial_objects,
   t_geometry_builder,
   t_vector_ge,
   t_action_on_forms,
   t_orbits,
   t_poset_classification_control,
};



typedef enum monomial_ordering_type monomial_ordering_type;
typedef enum diophant_equation_type diophant_equation_type;
typedef enum symbol_table_object_type symbol_table_object_type;


enum TokenType
{
   NONE,
   NAME,
   NUMBER,
   END,
   PLUS=´+´,
   MINUS=´-´,
   MULTIPLY=´*´,
   DIVIDE=´/´,
```

```
   ASSIGN=´=´,
   LHPAREN=´(´,
   RHPAREN=´)´,
   COMMA=´,´,
   NOT=´!´,

   // comparisons
   LT=´<´,
   GT=´>´,
   LE,     // <=
   GE,     // >=
   EQ,     // ==
   NE,     // !=
   AND,    // &&
   OR,      // ||

   // special assignments

   ASSIGN_ADD,  //  +=
   ASSIGN_SUB,  //  +-
   ASSIGN_MUL,  //  +*
   ASSIGN_DIV   //  +/

};

enum syntax_tree_node_operation_type
{
   operation_type_nothing,
   operation_type_mult,
   operation_type_add
};


enum data_input_stream_type {
   t_data_input_stream_unknown,
   t_data_input_stream_set_of_points,
   t_data_input_stream_set_of_lines,
   t_data_input_stream_set_of_points_and_lines,
   t_data_input_stream_set_of_packing,
   t_data_input_stream_file_of_points,
   t_data_input_stream_file_of_lines,
   t_data_input_stream_file_of_packings,
   t_data_input_stream_file_of_packings_through_spread_table,
   t_data_input_stream_file_of_point_set,
   t_data_input_stream_file_of_designs,
   t_data_input_stream_file_of_incidence_geometries,
   t_data_input_stream_file_of_incidence_geometries_by_row_ranks,
```

```
   t_data_input_stream_incidence_geometry,
   t_data_input_stream_incidence_geometry_by_row_ranks,
   t_data_input_stream_from_parallel_search,

};


enum CRC_type {
   t_CRC_16,
   t_CRC_32,
   t_CRC_771_30,

};




}}


#include "algebra/algebra.h"
#include "algebraic_geometry/algebraic_geometry.h"
#include "coding_theory/coding_theory.h"
#include "combinatorics/combinatorics.h"
#include "cryptography/cryptography.h"
#include "data_structures/data_structures.h"
#include "expression_parser/expression_parser.h"
#include "finite_fields/finite_fields.h"
#include "geometry/geometry.h"
#include "geometry_builder/geometry_builder.h"
#include "graph_theory/graph_theory.h"
#include "graph_theory_nauty/graph_theory_nauty.h"
#include "graphics/graphics.h"
#include "knowledge_base/knowledge_base.h"
#include "linear_algebra/linear_algebra.h"
#include "number_theory/number_theory.h"
#include "orbiter_kernel_system/orbiter_kernel_system.h"
#include "orthogonal/orthogonal.h"
#include "polish/polish.h"
#include "ring_theory/ring_theory.h"
#include "solvers/solvers.h"


// Eigen_interface:
void orbiter_eigenvalues(int *Mtx, int nb_points, double *E, int verbose_level);
```

```
#endif /* ORBITER_SRC_LIB_FOUNDATIONS_FOUNDATIONS_H_ */
```

3.2 Algebra

```
// algebra.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005


#ifndef ORBITER_SRC_LIB_FOUNDATIONS_ALGEBRA_AND_NUMBER_THEORY_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_ALGEBRA_AND_NUMBER_THEORY_H_



namespace orbiter {
namespace layer1_foundations {
namespace algebra {

// ##############################################################################
// a_domain.cpp
// ##############################################################################

enum domain_kind {
   not_applicable, domain_the_integers, domain_integer_fractions
};


//! related to the computation of Young representations


class a_domain {
public:
   domain_kind kind;
   int size_of_instance_in_int;

   a_domain();
   ~a_domain();

   void init_integers(int verbose_level);
   void init_integer_fractions(int verbose_level);
   int as_int(int *elt, int verbose_level);
   void make_integer(int *elt, int n, int verbose_level);
   void make_zero(int *elt, int verbose_level);
   void make_zero_vector(int *elt, int len, int verbose_level);
   int is_zero_vector(int *elt, int len, int verbose_level);
```

```
int is_zero(int *elt, int verbose_level);
void make_one(int *elt, int verbose_level);
int is_one(int *elt, int verbose_level);
void copy(int *elt_from, int *elt_to, int verbose_level);
void copy_vector(int *elt_from, int *elt_to,
   int len, int verbose_level);
void swap_vector(int *elt1, int *elt2, int n, int verbose_level);
void swap(int *elt1, int *elt2, int verbose_level);
void add(int *elt_a, int *elt_b, int *elt_c, int verbose_level);
void add_apply(int *elt_a, int *elt_b, int verbose_level);
void subtract(int *elt_a, int *elt_b, int *elt_c, int verbose_level);
void negate(int *elt, int verbose_level);
void negate_vector(int *elt, int len, int verbose_level);
void mult(int *elt_a, int *elt_b, int *elt_c, int verbose_level);
void mult_apply(int *elt_a, int *elt_b, int verbose_level);
void power(int *elt_a, int *elt_b, int n, int verbose_level);
void divide(int *elt_a, int *elt_b, int *elt_c, int verbose_level);
void inverse(int *elt_a, int *elt_b, int verbose_level);
void print(int *elt);
void print_vector(int *elt, int n);
void print_matrix(int *A, int m, int n);
void print_matrix_for_maple(int *A, int m, int n);
void make_element_from_integer(int *elt, int n, int verbose_level);
void mult_by_integer(int *elt, int n, int verbose_level);
void divide_by_integer(int *elt, int n, int verbose_level);
int *offset(int *A, int i);
int Gauss_echelon_form(int *A, int f_special, int f_complete,
   int *base_cols,
   int f_P, int *P, int m, int n, int Pn, int verbose_level);
   // returns the rank which is the number
   // of entries in base_cols
   // A is a m x n matrix,
   // P is a m x Pn matrix (if f_P is TRUE)
void Gauss_step(
      int *v1, int *v2, int len, int idx, int verbose_level);
   // afterwards: v2[idx] = 0 and v1,v2 span the same space as before
   // v1 is not changed if v1[idx] is nonzero
void matrix_get_kernel(
      int *M, int m, int n, int *base_cols,
   int nb_base_cols,
   int &kernel_m, int &kernel_n, int *kernel, int verbose_level);
   // kernel must point to the appropriate amount of memory!
   // (at least n * (n - nb_base_cols) int´s)
   // kernel is stored as column vectors,
   // i.e. kernel_m = n and kernel_n = n - nb_base_cols.
void matrix_get_kernel_as_row_vectors(
      int *M, int m, int n,
```

```
      int *base_cols, int nb_base_cols,
      int &kernel_m, int &kernel_n, int *kernel, int verbose_level);
      // kernel must point to the appropriate amount of memory!
      // (at least n * (n - nb_base_cols) int´s)
      // kernel is stored as row vectors,
      // i.e. kernel_m = n - nb_base_cols and kernel_n = n.
   void get_image_and_kernel(
         int *M, int n, int &rk, int verbose_level);
   void complete_basis(
         int *M, int m, int n, int verbose_level);
   void mult_matrix(
         int *A, int *B, int *C, int ma, int na, int nb,
      int verbose_level);
   void mult_matrix3(
         int *A, int *B, int *C, int *D, int n,
      int verbose_level);
   void add_apply_matrix(
         int *A, int *B, int m, int n,
      int verbose_level);
   void matrix_mult_apply_scalar(
         int *A, int *s, int m, int n,
      int verbose_level);
   void make_block_matrix_2x2(
         int *Mtx, int n, int k,
      int *A, int *B, int *C, int *D, int verbose_level);
      // A is k x k,
      // B is k x (n - k),
      // C is (n - k) x k,
      // D is (n - k) x (n - k),
      // Mtx is n x n
   void make_identity_matrix(
         int *A, int n, int verbose_level);
   void matrix_inverse(
         int *A, int *Ainv, int n, int verbose_level);
   void matrix_invert(
         int *A, int *T, int *basecols, int *Ainv, int n,
      int verbose_level);

};


// ##############################################################################
// algebra_global.cpp
// ##############################################################################

//! global functions related to finite fields, irreducible polynomials and such
```

```
class algebra_global {
public:
   void count_subprimitive(int Q_max, int H_max);
   void formula_subprimitive(int d, int q,
         ring_theory::longinteger_object &Rdq,
         int &g, int verbose_level);
   void formula(int d, int q,
         ring_theory::longinteger_object &Rdq,
         int verbose_level);
   int subprimitive(int q, int h);
   int period_of_sequence(int *v, int l);
   void subexponent(
         int q, int Q, int h, int f, int j, int k, int &s, int &c);
   const char *plus_minus_string(int epsilon);
   const char *plus_minus_letter(int epsilon);
   void display_all_PHG_elements(int n, int q);
   void test_unipoly(field_theory::finite_field *F);
   void test_unipoly2(field_theory::finite_field *F);
   int is_diagonal_matrix(int *A, int n);


   void test_longinteger();
   void test_longinteger2();
   void test_longinteger3();
   void test_longinteger4();
   void test_longinteger5();
   void test_longinteger6();
   void test_longinteger7();
   void test_longinteger8();
   void longinteger_collect_setup(int &nb_agos,
         ring_theory::longinteger_object *&agos,
         int *&multiplicities);
   void longinteger_collect_free(int &nb_agos,
         ring_theory::longinteger_object *&agos,
         int *&multiplicities);
   void longinteger_collect_add(int &nb_agos,
         ring_theory::longinteger_object *&agos,
         int *&multiplicities,
         ring_theory::longinteger_object &ago);
   void longinteger_collect_print(std::ostream &ost,
         int &nb_agos, ring_theory::longinteger_object *&agos,
         int *&multiplicities);
```

```
void order_of_q_mod_n(
      int q, int n_min, int n_max, int verbose_level);
void power_function_mod_n(
      int k, int n, int verbose_level);

void do_trace(
      field_theory::finite_field *F, int verbose_level);
void do_norm(
      field_theory::finite_field *F, int verbose_level);
void do_cheat_sheet_GF(
      field_theory::finite_field *F, int verbose_level);
void export_tables(
      field_theory::finite_field *F, int verbose_level);
void do_cheat_sheet_ring(
      ring_theory::homogeneous_polynomial_domain *HPD,
      int verbose_level);
void gl_random_matrix(
      field_theory::finite_field *F, int k, int verbose_level);

// functions with file based input:
void apply_Walsh_Hadamard_transform(
      field_theory::finite_field *F,
      std::string &fname_csv_in, int n, int verbose_level);
void algebraic_normal_form(
      field_theory::finite_field *F,
      int n,
      int *func, int len, int verbose_level);
void algebraic_normal_form_of_boolean_function(
      field_theory::finite_field *F,
      std::string &fname_csv_in, int n, int verbose_level);
void apply_trace_function(
      field_theory::finite_field *F,
      std::string &fname_csv_in, int verbose_level);
void apply_power_function(
      field_theory::finite_field *F,
      std::string &fname_csv_in, long int d, int verbose_level);
void identity_function(
      field_theory::finite_field *F,
      std::string &fname_csv_out, int verbose_level);
void Walsh_matrix(
      field_theory::finite_field *F,
      int n, int *&W, int verbose_level);
void Vandermonde_matrix(
      field_theory::finite_field *F,
      int *&W, int *&W_inv, int verbose_level);
void search_APN(
      field_theory::finite_field *F,
```

```
      int delta_max, int verbose_level);
void search_APN_recursion(
      field_theory::finite_field *F,
      int *f, int depth, int f_normalize,
      int &delta_max, int &nb_times,
      std::vector<std::vector<int> > &Solutions,
      int *A_matrix, int *B_matrix,
      int *Count_ab, int *nb_times_ab,
      int verbose_level);
int search_APN_perform_checks(field_theory::finite_field *F,
      int *f, int depth,
      int delta_max,
      int *A_matrix, int *B_matrix, int *Count_ab,
      int verbose_level);
void search_APN_undo_checks(field_theory::finite_field *F,
      int *f, int depth,
      int delta_max,
      int *A_matrix, int *B_matrix, int *Count_ab,
      int verbose_level);
int perform_single_check(field_theory::finite_field *F,
      int *f, int depth, int i, int delta_max,
      int *A_matrix, int *B_matrix, int *Count_ab,
      int verbose_level);
void undo_single_check(field_theory::finite_field *F,
      int *f, int depth, int i, int delta_max,
      int *A_matrix, int *B_matrix, int *Count_ab,
      int verbose_level);
void search_APN_old(
      field_theory::finite_field *F, int verbose_level);
void search_APN_recursion_old(
      field_theory::finite_field *F,
      int *f, int depth, int f_normalize,
      int &delta_min, int &nb_times,
      std::vector<std::vector<int> > &Solutions,
      int *nb_times_ab,
      int verbose_level);
int differential_uniformity(
      field_theory::finite_field *F,
      int *f, int *nb_times_ab, int verbose_level);
int differential_uniformity_with_fibre(
      field_theory::finite_field *F,
      int *f, int *nb_times_ab, int *&Fibre,
      int verbose_level);

void O4_isomorphism_4to2(
      field_theory::finite_field *F,
   int *At, int *As, int &f_switch, int *B,
```

```
      int verbose_level);
   void O4_isomorphism_2to4(
         field_theory::finite_field *F,
      int *At, int *As, int f_switch, int *B);
   void O4_grid_coordinates_rank(
         field_theory::finite_field *F,
      int x1, int x2, int x3, int x4,
      int &grid_x, int &grid_y, int verbose_level);
   void O4_grid_coordinates_unrank(
         field_theory::finite_field *F,
      int &x1, int &x2, int &x3, int &x4, int grid_x,
      int grid_y, int verbose_level);
   void O4_find_tangent_plane(
         field_theory::finite_field *F,
      int pt_x1, int pt_x2, int pt_x3, int pt_x4,
      int *tangent_plane, int verbose_level);
   void Nth_roots(
         field_theory::finite_field *F,
         int n, int verbose_level);

};



// ##############################################################################
// generators_symplectic_group.cpp
// ##############################################################################


//! generators of the symplectic group


class generators_symplectic_group {
public:

   field_theory::finite_field *F; // no ownership, do not destroy
   int n; // must be even
   int n_half; // n / 2
   int q;
   int qn; // = q^n

   int *nb_candidates; // [n + 1]
   int *cur_candidate; // [n]
   int **candidates; // [n + 1][q^n]

   int *Mtx; // [n * n]
   int *v; // [n]
```

```
   int *v2; // [n]
   int *w; // [n]
   int *Points; // [qn * n]

   int nb_gens;
   int *Data;
   int *transversal_length;

   generators_symplectic_group();
   ~generators_symplectic_group();
   void init(
         field_theory::finite_field *F, int n, int verbose_level);
   int count_strong_generators(int &nb, int *transversal_length,
      int &first_moved, int depth, int verbose_level);
   int get_strong_generators(int *Data, int &nb, int &first_moved,
      int depth, int verbose_level);
   void create_first_candidate_set(int verbose_level);
   void create_next_candidate_set(int level, int verbose_level);
   int dot_product(int *u1, int *u2);
};

// ##############################################################################
// gl_class_rep.cpp
// ##############################################################################

//! conjugacy class in GL(n,q) described using rational normal form

class gl_class_rep {
public:
   data_structures::int_matrix *type_coding;
   ring_theory::longinteger_object *centralizer_order;
   ring_theory::longinteger_object *class_length;

   gl_class_rep();
   ~gl_class_rep();
   void init(int nb_irred, int *Select_polynomial,
      int *Select_partition, int verbose_level);
   void print(int nb_irred,  int *Select_polynomial,
         int *Select_partition, int verbose_level);
   void compute_vector_coding(gl_classes *C, int &nb_irred,
      int *&Poly_degree, int *&Poly_mult, int *&Partition_idx,
      int verbose_level);
   void centralizer_order_Kung(gl_classes *C,
         ring_theory::longinteger_object &co,
      int verbose_level);
};
```

```
// ##############################################################################
// group_generators_domain.cpp
// ##############################################################################

//! generators for various classes of groups

class group_generators_domain {
public:
   group_generators_domain();
   ~group_generators_domain();
   void generators_symmetric_group(int deg,
      int &nb_perms, int *&perms, int verbose_level);
   void generators_cyclic_group(int deg,
      int &nb_perms, int *&perms, int verbose_level);
   void generators_dihedral_group(int deg,
      int &nb_perms, int *&perms, int verbose_level);
   void generators_dihedral_involution(int deg,
      int &nb_perms, int *&perms, int verbose_level);
   void generators_identity_group(int deg,
      int &nb_perms, int *&perms, int verbose_level);
   void generators_Hall_reflection(int nb_pairs,
         int &nb_perms, int *&perms, int &degree,
         int verbose_level);
   void generators_Hall_reflection_normalizer_group(int nb_pairs,
         int &nb_perms, int *&perms, int &degree,
         int verbose_level);
   void order_Hall_reflection_normalizer_factorized(int nb_pairs,
         int *&factors, int &nb_factors);
   void order_Bn_group_factorized(int n,
      int *&factors, int &nb_factors);
   void generators_Bn_group(int n, int &deg,
      int &nb_perms, int *&perms, int verbose_level);
   void generators_direct_product(int deg1, int nb_perms1, int *perms1,
      int deg2, int nb_perms2, int *perms2,
      int &deg3, int &nb_perms3, int *&perms3,
      int verbose_levels);
   void generators_concatenate(
         int deg1, int nb_perms1, int *perms1,
      int deg2, int nb_perms2, int *perms2,
      int &deg3, int &nb_perms3, int *&perms3,
      int verbose_level);
   int matrix_group_base_len_projective_group(int n, int q,
      int f_semilinear, int verbose_level);
   int matrix_group_base_len_affine_group(int n, int q,
      int f_semilinear, int verbose_level);
   int matrix_group_base_len_general_linear_group(int n, int q,
      int f_semilinear, int verbose_level);
```

```
void order_POmega_epsilon(int epsilon, int m, int q,
      ring_theory::longinteger_object &o, int verbose_level);
void order_PO_epsilon(int f_semilinear, int epsilon, int k, int q,
      ring_theory::longinteger_object &go, int verbose_level);
// k is projective dimension
void order_PO(int epsilon, int m, int q,
      ring_theory::longinteger_object &o,
   int verbose_level);
void order_Pomega(int epsilon, int k, int q,
      ring_theory::longinteger_object &go,
   int verbose_level);
void order_PO_plus(int m, int q,
      ring_theory::longinteger_object &o, int verbose_level);
void order_PO_minus(int m, int q,
      ring_theory::longinteger_object &o, int verbose_level);
// m = Witt index, the dimension is n = 2m+2
void order_PO_parabolic(int m, int q,
      ring_theory::longinteger_object &o, int verbose_level);
void order_Pomega_plus(int m, int q,
      ring_theory::longinteger_object &o, int verbose_level);
// m = Witt index, the dimension is n = 2m
void order_Pomega_minus(int m, int q,
      ring_theory::longinteger_object &o, int verbose_level);
// m = half the dimension,
// the dimension is n = 2m, the Witt index is m - 1
void order_Pomega_parabolic(
      int m, int q, ring_theory::longinteger_object &o,
   int verbose_level);
// m = Witt index, the dimension is n = 2m + 1
int index_POmega_in_PO(
      int epsilon, int m, int q, int verbose_level);


void diagonal_orbit_perm(
      int n, field_theory::finite_field *F,
      long int *orbit, long int *orbit_inv,
      int verbose_level);
void frobenius_orbit_perm(
      int n, field_theory::finite_field *F,
   long int *orbit, long int *orbit_inv,
   int verbose_level);
void projective_matrix_group_base_and_orbits(
      int n, field_theory::finite_field *F,
   int f_semilinear,
   int base_len, int degree,
   long int *base, int *transversal_length,
   long int **orbit, long int **orbit_inv,
```

```
   int verbose_level);
void projective_matrix_group_base_and_transversal_length(
      int n, field_theory::finite_field *F,
   int f_semilinear,
   int base_len, int degree,
   long int *base, int *transversal_length,
   int verbose_level);
void affine_matrix_group_base_and_transversal_length(
      int n, field_theory::finite_field *F,
   int f_semilinear,
   int base_len, int degree,
   long int *base, int *transversal_length,
   int verbose_level);
void general_linear_matrix_group_base_and_transversal_length(
      int n, field_theory::finite_field *F,
   int f_semilinear,
   int base_len, int degree,
   long int *base, int *transversal_length,
   int verbose_level);
void strong_generators_for_projective_linear_group(
   int n, field_theory::finite_field *F,
   int f_semilinear,
   int *&data, int &size, int &nb_gens,
   int verbose_level);
void strong_generators_for_affine_linear_group(
   int n, field_theory::finite_field *F,
   int f_semilinear,
   int *&data, int &size, int &nb_gens,
   int verbose_level);
void strong_generators_for_general_linear_group(
   int n, field_theory::finite_field *F,
   int f_semilinear,
   int *&data, int &size, int &nb_gens,
   int verbose_level);
void generators_for_parabolic_subgroup(
   int n, field_theory::finite_field *F,
   int f_semilinear, int k,
   int *&data, int &size, int &nb_gens,
   int verbose_level);
void generators_for_stabilizer_of_three_collinear_points_in_PGL4(
   int f_semilinear, field_theory::finite_field *F,
   int *&data, int &size, int &nb_gens,
   int verbose_level);
void generators_for_stabilizer_of_triangle_in_PGL4(
   int f_semilinear, field_theory::finite_field *F,
   int *&data, int &size, int &nb_gens,
   int verbose_level);
```

```
void builtin_transversal_rep_GLnq(int *A, int n,
      field_theory::finite_field *F,
   int f_semilinear, int i, int j, int verbose_level);
void affine_translation(
      int n, field_theory::finite_field *F,
      int coordinate_idx,
      int field_base_idx, int *perm,
      int verbose_level);
   // perm points to q^n int´s
   // field_base_idx is the base element whose
   // translation we compute, 0 \le field_base_idx < e
   // coordinate_idx is the coordinate in which we shift,
   // 0 \le coordinate_idx < n
void affine_multiplication(
      int n, field_theory::finite_field *F,
   int multiplication_order, int *perm,
   int verbose_level);
   // perm points to q^n int´s
   // compute the diagonal multiplication by alpha, i.e.
   // the multiplication by alpha of each component
void affine_frobenius(
      int n, field_theory::finite_field *F,
      int k, int *perm,
      int verbose_level);
   // perm points to q^n int´s
   // compute the diagonal action of the Frobenius
   // automorphism to the power k, i.e.,
   // raises each component to the p^k-th power
int all_affine_translations_nb_gens(
      int n, field_theory::finite_field *F);
void all_affine_translations(
      int n, field_theory::finite_field *F, int *gens);
void affine_generators(
      int n, field_theory::finite_field *F,
      int f_translations,
      int f_semilinear, int frobenius_power,
      int f_multiplication, int multiplication_order,
      int &nb_gens, int &degree, int *&gens,
      int &base_len, long int *&the_base,
      int verbose_level);
void PG_element_modified_not_in_subspace_perm(
      field_theory::finite_field *F,
      int n, int m,
   long int *orbit, long int *orbit_inv,
   int verbose_level);
```

```
};


// ##############################################################################
// gl_classes.cpp
// ##############################################################################

//! to list all conjugacy classes in GL(n,q)

class gl_classes {
public:
   int k;
   int q;
   field_theory::finite_field *F;
   ring_theory::table_of_irreducible_polynomials *Table_of_polynomials;
   int *Nb_part;
   int **Partitions;
   int *v, *w; // [k], used in choose_basis_for_rational_normal_form_block

   gl_classes();
   ~gl_classes();
   void init(int k,
         field_theory::finite_field *F, int verbose_level);
   int select_partition_first(
         int *Select, int *Select_partition,
      int verbose_level);
   int select_partition_next(
         int *Select, int *Select_partition,
      int verbose_level);
   int first(
         int *Select, int *Select_partition, int verbose_level);
   int next(
         int *Select, int *Select_partition, int verbose_level);
   void make_matrix_from_class_rep(int *Mtx, gl_class_rep *R,
      int verbose_level);
   void make_matrix_in_rational_normal_form(
         int *Mtx, int *Select, int *Select_Partition,
         int verbose_level);
   void centralizer_order_Kung_basic(
         int nb_irreds,
      int *poly_degree, int *poly_mult, int *partition_idx,
      ring_theory::longinteger_object &co,
      int verbose_level);
   void centralizer_order_Kung(
         int *Select_polynomial,
      int *Select_partition,
      ring_theory::longinteger_object &co,
```

```
   int verbose_level);
   // Computes the centralizer order of a matrix in GL(k,q)
   // according to Kung´s formula~\cite{Kung81}.
void make_classes(
      gl_class_rep *&R, int &nb_classes,
   int f_no_eigenvalue_one, int verbose_level);
void identify_matrix(int *Mtx, gl_class_rep *R, int *Basis,
   int verbose_level);
void identify2(int *Mtx,
      ring_theory::unipoly_object &poly, int *Mult,
   int *Select_partition, int *Basis, int verbose_level);
void compute_generalized_kernels_for_each_block(
   int *Mtx, int *Irreds, int nb_irreds,
   int *Degree, int *Mult, matrix_block_data *Data,
   int verbose_level);
void compute_generalized_kernels(
      matrix_block_data *Data, int *M2,
   int d, int b0, int m, int *poly_coeffs, int verbose_level);
int identify_partition(int *part, int m, int verbose_level);
void choose_basis_for_rational_normal_form(int *Mtx,
   matrix_block_data *Data, int nb_irreds,
   int *Basis,
   int verbose_level);
void choose_basis_for_rational_normal_form_block(
      int *Mtx,
   matrix_block_data *Data,
   int *Basis, int &b,
   int verbose_level);
void generators_for_centralizer(
      int *Mtx, gl_class_rep *R,
   int *Basis, int **&Gens, int &nb_gens, int &nb_alloc,
   int verbose_level);
void centralizer_generators(
      int *Mtx,
      ring_theory::unipoly_object &poly,
   int *Mult, int *Select_partition,
   int *Basis, int **&Gens, int &nb_gens, int &nb_alloc,
   int verbose_level);
void centralizer_generators_block(
      int *Mtx,
      matrix_block_data *Data,
   int nb_irreds, int h,
   int **&Gens, int &nb_gens, int &nb_alloc,
   int verbose_level);
int choose_basis_for_rational_normal_form_coset(int level1,
   int level2, int &coset,
   int *Mtx, matrix_block_data *Data, int &b, int *Basis,
```

```
      int verbose_level);
   int find_class_rep(gl_class_rep *Reps, int nb_reps,
      gl_class_rep *R, int verbose_level);
   void report(std::ostream &ost, int verbose_level);
   void print_matrix_and_centralizer_order_latex(
         std::ostream &ost,
      gl_class_rep *R);
};



// ##############################################################################
// heisenberg.cpp
// ##############################################################################

//! Heisenberg group of n x n matrices


class heisenberg {

public:
   int q;
   field_theory::finite_field *F;
   int n;
   int len; // 2 * n + 1
   int group_order; // q^len

   int *Elt1;
   int *Elt2;
   int *Elt3;
   int *Elt4;

   heisenberg();
   ~heisenberg();
   void init(field_theory::finite_field *F, int n, int verbose_level);
   void unrank_element(int *Elt, long int rk);
   long int rank_element(int *Elt);
   void element_add(int *Elt1, int *Elt2, int *Elt3, int verbose_level);
   void element_negate(int *Elt1, int *Elt2, int verbose_level);
   int element_add_by_rank(int rk_a, int rk_b, int verbose_level);
   int element_negate_by_rank(int rk_a, int verbose_level);
   void group_table(int *&Table, int verbose_level);
   void group_table_abv(int *&Table_abv, int verbose_level);
   void generating_set(int *&gens, int &nb_gens, int verbose_level);


};
```

```
// ##############################################################################
// interface_gap_low.cpp:
// ##############################################################################

//! interface to GAP at the foundation level

class interface_gap_low {
public:

   interface_gap_low();
   ~interface_gap_low();
   void fining_set_stabilizer_in_collineation_group(
         field_theory::finite_field *F,
         int d, long int *Pts, int nb_pts,
         std::string &fname,
         int verbose_level);
   void collineation_set_stabilizer(
         std::ostream &ost,
         field_theory::finite_field *F,
         int d, long int *Pts, int nb_pts,
         int verbose_level);
   void write_matrix(
         std::ostream &ost,
         field_theory::finite_field *F,
         int *Mtx, int d,
         int verbose_level);
   void write_element_of_finite_field(
         std::ostream &ost,
         field_theory::finite_field *F, int a);

};

// ##############################################################################
// interface_magma_low.cpp:
// ##############################################################################

//! interface to magma at the foundation level

class interface_magma_low {
public:

   interface_magma_low();
   ~interface_magma_low();
   void magma_set_stabilizer_in_collineation_group(
         field_theory::finite_field *F,
```

```
        int d, long int *Pts, int nb_pts,
        std::string &fname,
        int verbose_level);

};


// ##############################################################################
// matrix_block_data.cpp
// ##############################################################################

//! rational normal form of a matrix in GL(n,q) for gl_class_rep

class matrix_block_data {
public:
   int d;
   int m;
   int *poly_coeffs;
   int b0;
   int b1;

   data_structures::int_matrix *K;
   int cnt;
   int *dual_part;
   int *part;
   int height;
   int part_idx;

   matrix_block_data();
   ~matrix_block_data();
   void allocate(int k);
};

// ##############################################################################
// null_polarity_generator.cpp:
// ##############################################################################

//! construction of all null polarities

class null_polarity_generator {
public:

   field_theory::finite_field *F; // no ownership, do not destroy
   int n, q;
   int qn; // = q^n

   int *nb_candidates; // [n + 1]
```

```
   int *cur_candidate; // [n]
   int **candidates; // [n + 1][q^n]

   int *Mtx; // [n * n]
   int *v; // [n]
   int *w; // [n]
   int *Points; // [qn * n]

   int nb_gens;
   int *Data;
   int *transversal_length;

   null_polarity_generator();
   ~null_polarity_generator();
   void init(
         field_theory::finite_field *F,
         int n, int verbose_level);
   int count_strong_generators(int &nb, int *transversal_length,
      int &first_moved, int depth, int verbose_level);
   int get_strong_generators(int *Data, int &nb, int &first_moved,
      int depth, int verbose_level);
   void backtrack_search(int &nb_sol, int depth, int verbose_level);
   void create_first_candidate_set(int verbose_level);
   void create_next_candidate_set(int level, int verbose_level);
   int dot_product(int *u1, int *u2);
};


// ##############################################################################
// rank_checker.cpp:
// ##############################################################################


//! to check whether any d - 1 elements of a given set are linearly independent


class rank_checker {

public:
   field_theory::finite_field *GFq;
   int m, n, d;

   int *M1; // [m * n]
   int *M2; // [m * n]
   int *base_cols; // [n]
   int *set; // [n] used in check_mindist
```

```
   rank_checker();
   ~rank_checker();
   void init(
         field_theory::finite_field *GFq,
         int m, int n, int d);
   int check_rank(
         int len, long int *S, int verbose_level);
   int check_rank_matrix_input(int len, long int *S, int dim_S,
      int verbose_level);
   int check_rank_last_two_are_fixed(
         int len, long int *S, int verbose_level);
   int compute_rank_row_vectors(
         int len, long int *S, int f_projective, int verbose_level);
};



// ##############################################################################
// vector_space.cpp:
// ##############################################################################


//! finite dimensional vector space over a finite field


class vector_space {
public:

   int dimension;
   field_theory::finite_field *F;

   long int (*rank_point_func)(int *v, void *data);
   void (*unrank_point_func)(int *v, long int rk, void *data);
   void *rank_point_data;
   int *v1; // [dimension]
   int *base_cols; // [dimension]
   int *base_cols2; // [dimension]
   int *M1; // [dimension * dimension]
   int *M2; // [dimension * dimension]


   vector_space();
   ~vector_space();
   void init(field_theory::finite_field *F, int dimension,
         int verbose_level);
   void init_rank_functions(
      long int (*rank_point_func)(int *v, void *data),
```

```
      void (*unrank_point_func)(int *v, long int rk, void *data),
      void *data,
      int verbose_level);
   void unrank_basis(int *Mtx, long int *set, int len);
   void rank_basis(int *Mtx, long int *set, int len);
   void unrank_point(int *v, long int rk);
   long int rank_point(int *v);
   int RREF_and_rank(int *basis, int k);
   int is_contained_in_subspace(int *v, int *basis, int k);
   int compare_subspaces_ranked(
         long int *set1, long int *set2, int k, int verbose_level);
      // equality test for subspaces given by ranks of basis elements
};



}}}


#endif /* ORBITER_SRC_LIB_FOUNDATIONS_ALGEBRA_AND_NUMBER_THEORY_H_ */
```

3.3 Algebraic Geometry

```
/*
 * surfaces.h
 *
 *  Created on: Jul 29, 2020
 *      Author: betten
 */

#ifndef SRC_LIB_FOUNDATIONS_ALGEBRAIC_GEOMETRY_ALGEBRAIC_GEOMETRY_H_
#define SRC_LIB_FOUNDATIONS_ALGEBRAIC_GEOMETRY_ALGEBRAIC_GEOMETRY_H_

namespace orbiter {
namespace layer1_foundations {
namespace algebraic_geometry {




// ##############################################################################
// algebraic_geometry_global.cpp
// ##############################################################################

//! catch all class for everything related to algebraic geometry


class algebraic_geometry_global {

public:

   algebraic_geometry_global();
   ~algebraic_geometry_global();
   void analyze_del_Pezzo_surface(
         geometry::projective_space *P,
         expression_parser::formula *Formula,
         std::string &evaluate_text,
         int verbose_level);
   void report_grassmannian(
         geometry::projective_space *P,
         int k,
         int verbose_level);
   void map(
         geometry::projective_space *P,
         std::string &ring_label,
         std::string &formula_label,
         std::string &evaluate_text,
         long int *&Image_pts,
         int &N_points,
```

```
        int verbose_level);


};



// ##############################################################################
// arc_lifting_with_two_lines.cpp
// ##############################################################################

//! creates a cubic surface from a 6-arc in a plane


class arc_lifting_with_two_lines {

public:

   int q;
   field_theory::finite_field *F; // do not free

   surface_domain *Surf; // do not free

   long int *Arc6;
   int arc_size; // = 6

   long int line1, line2;

   long int plane_rk;

   int *Arc_coords; // [6 * 4]

   long int P[6];

   long int transversal_01;
   long int transversal_23;
   long int transversal_45;

   long int transversal[4];

   long int input_Lines[9];

   int coeff[20];
   long int lines27[27];

   arc_lifting_with_two_lines();
   ~arc_lifting_with_two_lines();
```

```
   void create_surface(
      surface_domain *Surf,
      long int *Arc6, long int line1, long int line2,
      int verbose_level);
   // The arc must be given as points in PG(3,q), not in PG(2,q).
};



// ##############################################################################
// clebsch_map.cpp
// ##############################################################################

//! records the images of a specific Clebsch map, used by class layer5_applicatio
ns::applications_in_algebraic_geometry::cubic_surfaces_in_general::surface_clebsc
h_map


class clebsch_map {

public:
   surface_domain *Surf;
   surface_object *SO;
   field_theory::finite_field *F;

   int hds, ds, ds_row;

   int line1, line2;
   int transversal;
   int tritangent_plane_idx;

   int line_idx[2];
   long int plane_rk_global;

   int intersection_points[6];
   int intersection_points_local[6];
   int Plane[16];
   int base_cols[4];


   long int *Clebsch_map; // [SO->nb_pts], = Image_rk
   int *Clebsch_coeff; // [SO->nb_pts * 4], = Image_coeff

   long int Arc[6];
   long int Blown_up_lines[6];

   clebsch_map();
```

```
   ~clebsch_map();
   void init_half_double_six(surface_object *SO,
         int hds, int verbose_level);
   void compute_Clebsch_map_down(int verbose_level);
   int compute_Clebsch_map_down_worker(
      long int *Image_rk, int *Image_coeff,
      int verbose_level);
   // assuming:
   // In:
   // SO->Lines[27]
   // SO->Pts[SO->nb_pts]
   // Out:
   // Image_rk[nb_pts]  (image point in the plane in local coordinates)
   //   Note Image_rk[i] is -1 if Pts[i] does not have an image.
   // Image_coeff[nb_pts * 4] (image point in the plane in PG(3,q) coordinates)
   void clebsch_map_print_fibers();
   void clebsch_map_find_arc_and_lines(int verbose_level);
   void report(std::ostream &ost, int verbose_level);

};


// ##############################################################################
// cubic_curve.cpp
// ##############################################################################

//! cubic curves in PG(2,q)


class cubic_curve {

public:
   int q;
   field_theory::finite_field *F;
   geometry::projective_space *P; // PG(2,q)


   int nb_monomials;


   ring_theory::homogeneous_polynomial_domain *Poly;
      // cubic polynomials in three variables
   ring_theory::homogeneous_polynomial_domain *Poly2;
      // quadratic polynomials in three variables

   ring_theory::partial_derivative *Partials;
```

```
   int *gradient; // 3 * Poly2->nb_monomials


   cubic_curve();
   ~cubic_curve();
   void init(
         field_theory::finite_field *F, int verbose_level);
   int compute_system_in_RREF(
         int nb_pts, long int *pt_list, int verbose_level);
   void compute_gradient(
         int *eqn_in, int verbose_level);
   void compute_singular_points(
         int *eqn_in,
         long int *Pts_on_curve, int nb_pts_on_curve,
         long int *Pts, int &nb_pts,
         int verbose_level);
   void compute_inflexion_points(
         int *eqn_in,
         long int *Pts_on_curve, int nb_pts_on_curve,
         long int *Pts, int &nb_pts,
         int verbose_level);

};




// ##############################################################################
// del_pezzo_surface_of_degree_two_domain.cpp
// ##############################################################################

//! domain for del Pezzo surfaces of degree two


class del_pezzo_surface_of_degree_two_domain {

public:
   field_theory::finite_field *F;
   geometry::projective_space *P3;
   geometry::projective_space *P2;
   geometry::grassmann *Gr; // Gr_{4,2}
   geometry::grassmann *Gr3; // Gr_{4,3}
   long int nb_lines_PG_3;
   ring_theory::homogeneous_polynomial_domain *Poly4_3;
      // quartic polynomials in three variables

   del_pezzo_surface_of_degree_two_domain();
```

```
   ~del_pezzo_surface_of_degree_two_domain();
   void init(
         geometry::projective_space *P3,
         ring_theory::homogeneous_polynomial_domain *Poly4_3,
         int verbose_level);
   void enumerate_points(int *coeff,
         std::vector<long int> &Pts,
         int verbose_level);
   void print_equation_with_line_breaks_tex(
         std::ostream &ost, int *coeffs);
   void unrank_point(int *v, long int rk);
   long int rank_point(int *v);

};


// ##############################################################################
// del_pezzo_surface_of_degree_two_object.cpp
// ##############################################################################

//! a del Pezzo surface of degree two


class del_pezzo_surface_of_degree_two_object {

public:
   del_pezzo_surface_of_degree_two_domain *Dom;

   expression_parser::formula *RHS;
   expression_parser::syntax_tree_node **Subtrees;
   int *Coefficient_vector;

   geometry::points_and_lines *pal;


   del_pezzo_surface_of_degree_two_object();
   ~del_pezzo_surface_of_degree_two_object();
   void init(
         del_pezzo_surface_of_degree_two_domain *Dom,
         expression_parser::formula *RHS,
         expression_parser::syntax_tree_node **Subtrees,
         int *Coefficient_vector,
         int verbose_level);
   void enumerate_points_and_lines(int verbose_level);
   void create_latex_report(std::string &label,
         std::string &label_tex, int verbose_level);
   void report_properties(
```

```
        std::ostream &ost, int verbose_level);
   void print_equation(std::ostream &ost);
   void print_points(std::ostream &ost);
   void print_all_points_on_surface(std::ostream &ost);
   void print_lines(std::ostream &ost);

};



// ##############################################################################
// eckardt_point_info.cpp
// ##############################################################################

//! information about the Eckardt points of a surface derived from a six-arc


class eckardt_point_info {

public:

   //surface_domain *Surf;
   geometry::projective_space *P2;
   long int arc6[6];

   int *bisecants; // [15]
   int *Intersections; // [15 * 15]
   int *B_pts; // [nb_B_pts]
   int *B_pts_label; // [nb_B_pts * 3]
   int nb_B_pts; // at most 15
   int *E2; // [6 * 5 * 2] Eckardt points of the second type
   int nb_E2; // at most 30
   int *conic_coefficients; // [6 * 6]
   eckardt_point *E;
   int nb_E;

   eckardt_point_info();
   ~eckardt_point_info();
   void init(geometry::projective_space *P2,
        long int *arc6, int verbose_level);
   void print_bisecants(std::ostream &ost, int verbose_level);
   void print_intersections(std::ostream &ost, int verbose_level);
   void print_conics(std::ostream &ost, int verbose_level);
   void print_Eckardt_points(
        std::ostream &ost, int verbose_level);

};
```

```
// ##############################################################################
// eckardt_point.cpp
// ##############################################################################

//! Eckardt point on a cubic surface using the Schlaefli labeling


class eckardt_point {

public:

   int len;
   int pt;
   int index[3];


   eckardt_point();
   ~eckardt_point();
   void print();
   void latex(std::ostream &ost);
   void latex_index_only(std::ostream &ost);
   void latex_to_string(std::string &s);
   void latex_to_str_without_E(std::string &s);
   void init2(int i, int j);
   void init3(int ij, int kl, int mn);
   void init6(int i, int j, int k, int l, int m, int n);
   void init_by_rank(int rk);
   void three_lines(surface_domain *S, int *three_lines);
   int rank();
   void unrank(int rk,
         int &i, int &j, int &k, int &l, int &m, int &n);

};

// ##############################################################################
// quartic_curve_domain.cpp
// ##############################################################################

//! domain for quartic curves in PG(2,q) with 28 bitangents


class quartic_curve_domain {

public:
   field_theory::finite_field *F;
```

```
geometry::projective_space *P;

// we use the monomial ordering t_PART in all polynomial rings:

ring_theory::homogeneous_polynomial_domain *Poly1_3;
   // linear polynomials in three variables
ring_theory::homogeneous_polynomial_domain *Poly2_3;
   // quadratic polynomials in three variables
ring_theory::homogeneous_polynomial_domain *Poly3_3;
   // cubic polynomials in three variables
ring_theory::homogeneous_polynomial_domain *Poly4_3;
   // quartic polynomials in three variables

ring_theory::homogeneous_polynomial_domain *Poly3_4;
   // cubic polynomials in four variables

ring_theory::partial_derivative *Partials; // [3]

algebraic_geometry::schlaefli_labels *Schlaefli;

quartic_curve_domain();
~quartic_curve_domain();
void init(
      field_theory::finite_field *F,
      int verbose_level);
void init_polynomial_domains(int verbose_level);
void print_equation_maple(
      std::stringstream &ost, int *coeffs);
void print_equation_with_line_breaks_tex(
      std::ostream &ost, int *coeffs);
void print_gradient_with_line_breaks_tex(
      std::ostream &ost, int *coeffs);
void unrank_point(int *v, long int rk);
long int rank_point(int *v);
void unrank_line_in_dual_coordinates(int *v, long int rk);
void print_lines_tex(
      std::ostream &ost, long int *Lines, int nb_lines);
void compute_points_on_lines(
      long int *Pts, int nb_points,
      long int *Lines, int nb_lines,
      data_structures::set_of_sets *&pts_on_lines,
      int *&f_is_on_line,
      int verbose_level);
void multiply_conic_times_conic(int *six_coeff_a,
   int *six_coeff_b, int *fifteen_coeff,
   int verbose_level);
void multiply_conic_times_line(int *six_coeff,
```

```
      int *three_coeff, int *ten_coeff,
      int verbose_level);
   void multiply_line_times_line(int *line1,
      int *line2, int *six_coeff,
      int verbose_level);
   void multiply_three_lines(int *line1, int *line2, int *line3,
      int *ten_coeff,
      int verbose_level);
   void multiply_four_lines(int *line1, int *line2, int *line3, int *line4,
      int *fifteen_coeff,
      int verbose_level);
   void assemble_cubic_surface(int *f1, int *f2, int *f3, int *eqn20,
      int verbose_level);
   void create_surface(quartic_curve_object *Q, int *eqn20, int verbose_level);
   // Given a quartic Q in X1,X2,X3, compute an associated cubic surface
   // whose projection from (1,0,0,0) gives back the quartic Q.
   // Pick 4 bitangents L0,L1,L2,L3 so that the 8 points of tangency lie on a coni
c C.
   // Then, create the cubic surface with equation
   // (- lambda * mu) / 4 * X0^2 * L0 (the equation of the first of the four bitan
gents)
   // + X0 * lambda * C (the conic equation)
   // + L1 * L2 * L3 (the product of the equations of the last three bitangents)
   // Here 1, lambda, mu are the coefficients of a linear dependency between
   // Q (the quartic), C^2, L0*L1*L2*L3, so
   // Q + lambda * C^2 + mu * L0*L1*L2*L3 = 0.
   void compute_gradient(int *equation15, int *&gradient, int verbose_level);

};


// ##############################################################################
// quartic_curve_object_properties.cpp
// ##############################################################################

//! properties of a particular quartic curve surface in PG(2,q), as defined by an
 object of class quartic_curve_object


class quartic_curve_object_properties {

public:

   quartic_curve_object *Q0;


   data_structures::set_of_sets *pts_on_lines;
```

```
   // points are stored as indices into Pts[]
int *f_is_on_line; // [QO->nb_pts]

data_structures::tally *Bitangent_line_type;
int line_type_distribution[3];

data_structures::set_of_sets *lines_on_point;
data_structures::tally *Point_type;

int f_fullness_has_been_established;
int f_is_full;
int nb_Kovalevski;
int nb_Kovalevski_on;
int nb_Kovalevski_off;
int *Kovalevski_point_idx;
long int *Kovalevski_points;

long int *Pts_off;
int nb_pts_off;

data_structures::set_of_sets *pts_off_on_lines;
int *f_is_on_line2; // [QO->nb_pts]

data_structures::set_of_sets *lines_on_points_off;
data_structures::tally *Point_off_type;


int *gradient;

long int *singular_pts;
int nb_singular_pts;
int nb_non_singular_pts;

long int *tangent_line_rank_global; // [QO->nb_pts]
long int *tangent_line_rank_dual; // [nb_non_singular_pts]



quartic_curve_object_properties();
~quartic_curve_object_properties();
void init(quartic_curve_object *QO, int verbose_level);
void create_summary_file(std::string &fname,
      std::string &surface_label,
      std::string &col_postfix,
      int verbose_level);
void report_properties_simple(std::ostream &ost, int verbose_level);
void print_equation(std::ostream &ost);
```

```
   void print_gradient(std::ostream &ost);
   void print_general(std::ostream &ost);
   void print_points(std::ostream &ost);
   void print_all_points(std::ostream &ost);
   void print_bitangents(std::ostream &ost);
   void print_lines_with_points_on_them(std::ostream &ost,
         long int *Lines, int nb_lines,
         data_structures::set_of_sets *SoS);
   //void print_bitangents_with_points_on_them(std::ostream &ost);
   void points_on_curve_on_lines(int verbose_level);
   void report_bitangent_line_type(std::ostream &ost);
   void compute_gradient(int verbose_level);
   void compute_singular_points_and_tangent_lines(int verbose_level);
   // a singular point is a point where all partials vanish
   // We compute the set of singular points into Pts[nb_pts]

};



// ##############################################################################
// quartic_curve_object.cpp
// ##############################################################################

//! a particular quartic curve in PG(2,q), given by its equation


class quartic_curve_object {

public:
   int q;
   field_theory::finite_field *F;
   quartic_curve_domain *Dom;

   long int *Pts; // in increasing order
   int nb_pts;


   //long int *Lines;
   //int nb_lines;

   int eqn15[15];

   int f_has_bitangents;
   long int bitangents28[28];

   quartic_curve_object_properties *QP;
```

```
   quartic_curve_object();
   ~quartic_curve_object();
   void init_equation_but_no_bitangents(
         quartic_curve_domain *Dom,
         int *eqn15,
         int verbose_level);
   void init_equation_and_bitangents(
         quartic_curve_domain *Dom,
         int *eqn15, long int *bitangents28,
         int verbose_level);
   void init_equation_and_bitangents_and_compute_properties(
         quartic_curve_domain *Dom,
         int *eqn15, long int *bitangents28,
         int verbose_level);
   void enumerate_points(int verbose_level);
   void compute_properties(int verbose_level);
   void recompute_properties(int verbose_level);
   void identify_lines(long int *lines, int nb_lines, int *line_idx,
      int verbose_level);
   int find_point(long int P, int &idx);

};



// ##############################################################################
// schlaefli_labels.cpp
// ##############################################################################

//! schlaefli labeling of objects in cubic surfaces with 27 lines


class schlaefli_labels {

public:

   long int *Sets; // [30 * 2]
   int *M; // [6 * 6]
   long int *Sets2; // [15 * 2]


   std::string *Line_label; // [27]
   std::string *Line_label_tex; // [27]
```

```
   schlaefli_labels();
   ~schlaefli_labels();
   void init(int verbose_level);
};


// ##############################################################################
// schlaefli.cpp
// ##############################################################################

//! schlaefli labeling of objects in cubic surfaces with 27 lines


class schlaefli {

public:

   surface_domain *Surf;


   schlaefli_labels *Labels;


   int *Trihedral_pairs; // [nb_trihedral_pairs * 9]
   std::string *Trihedral_pair_labels; // [nb_trihedral_pairs]
   int *Trihedral_pairs_row_sets; // [nb_trihedral_pairs * 3]
   int *Trihedral_pairs_col_sets; // [nb_trihedral_pairs * 3]
   int nb_trihedral_pairs; // = 120

   int *Triads;
   int nb_triads; // = 40

   data_structures::tally *Classify_trihedral_pairs_row_values;
   data_structures::tally *Classify_trihedral_pairs_col_values;

   int nb_Eckardt_points; // = 45
   eckardt_point *Eckardt_points;

   std::string *Eckard_point_label; // [nb_Eckardt_points]
   std::string *Eckard_point_label_tex; // [nb_Eckardt_points]

   int nb_trihedral_to_Eckardt; // nb_trihedral_pairs * 6
   long int *Trihedral_to_Eckardt;
      // [nb_trihedral_pairs * 6]
      // first the three rows, then the three columns
      // long int so that we can induce the action on it
```

```
int nb_collinear_Eckardt_triples;
   // nb_trihedral_pairs * 2
int *collinear_Eckardt_triples_rank;
   // as three subsets of 45 = nb_Eckardt_points

data_structures::tally *Classify_collinear_Eckardt_triples;


// Schlaefli stuff, part2:


long int *Double_six; // [36 * 12]
std::string *Double_six_label_tex; // [36]


long int *Half_double_sixes; // [72 * 6]
   // warning: the half double sixes are sorted individually,
   // so the pairing between the lines
   // in the associated double six is gone.
std::string *Half_double_six_label_tex; // [72]

int *Half_double_six_to_double_six; // [72]
int *Half_double_six_to_double_six_row; // [72]


// Schlaefli stuff, part3:

int *adjacency_matrix_of_lines;
   // [27 * 27]
   // indexed by the lines in Schlaefli labeling

int *incidence_lines_vs_tritangent_planes;
   // [27 * 45]
   // indexed by the lines and tritangent planes in Schlaefli labeling

long int *Lines_in_tritangent_planes;
   // [45 * 3]
   // long int so that we can induce the action on it



schlaefli();
~schlaefli();
void init(surface_domain *Surf, int verbose_level);
```

```
void init_line_data(int verbose_level);
void init_Schlaefli_labels(int verbose_level);
void find_tritangent_planes_intersecting_in_a_line(
   int line_idx,
   int &plane1, int &plane2, int verbose_level);
void make_triads(int verbose_level);
void make_trihedral_pair_disjointness_graph(
      int *&Adj, int verbose_level);
void make_trihedral_pairs(int verbose_level);
void process_trihedral_pairs(int verbose_level);
int line_ai(int i);
int line_bi(int i);
int line_cij(int i, int j);
int type_of_line(int line);
   // 0 = a_i, 1 = b_i, 2 = c_ij
void index_of_line(int line, int &i, int &j);
   // returns i for a_i, i for b_i and (i,j) for c_ij
int third_line_in_tritangent_plane(int l1, int l2, int verbose_level);
void make_Tijk(int *T, int i, int j, int k);
void make_Tlmnp(int *T, int l, int m, int n, int p);
void make_Tdefght(int *T, int d, int e, int f, int g, int h, int t);
void make_Eckardt_points(int verbose_level);
void init_Trihedral_to_Eckardt(int verbose_level);
int Eckardt_point_from_tritangent_plane(int *tritangent_plane);
void init_collinear_Eckardt_triples(int verbose_level);
void find_trihedral_pairs_from_collinear_triples_of_Eckardt_points(
   int *E_idx, int nb_E,
   int *&T_idx, int &nb_T, int verbose_level);
void init_double_sixes(int verbose_level);
void create_half_double_sixes(int verbose_level);
int find_half_double_six(long int *half_double_six);
void ijklm2n(int i, int j, int k, int l, int m, int &n);
void ijkl2mn(int i, int j, int k, int l, int &m, int &n);
void ijk2lmn(int i, int j, int k, int &l, int &m, int &n);
void ij2klmn(int i, int j, int &k, int &l, int &m, int &n);
void get_half_double_six_associated_with_Clebsch_map(
   int line1, int line2, int transversal,
   int hds[6],
   int verbose_level);
void prepare_clebsch_map(int ds, int ds_row, int &line1,
   int &line2, int &transversal, int verbose_level);
void init_adjacency_matrix_of_lines(int verbose_level);
void init_incidence_matrix_of_lines_vs_tritangent_planes(
      int verbose_level);
void set_adjacency_matrix_of_lines(int i, int j);
int get_adjacency_matrix_of_lines(int i, int j);
int choose_tritangent_plane_for_Clebsch_map(
```

```
        int line_a, int line_b,
           int transversal_line, int verbose_level);

   void latex_table_of_double_sixes(std::ostream &ost);
   void latex_double_six_symbolic(std::ostream &ost, int idx);
   void latex_double_six_index_set(std::ostream &ost, int idx);
   void latex_table_of_half_double_sixes(std::ostream &ost);
   void print_Steiner_and_Eckardt(std::ostream &ost);
   void latex_abstract_trihedral_pair(std::ostream &ost, int t_idx);
   void latex_table_of_Schlaefli_labeling_of_lines(std::ostream &ost);
   void latex_trihedral_pair(std::ostream &ost, int *T, long int *TE);
   void latex_table_of_trihedral_pairs(std::ostream &ost);
   void latex_triads(std::ostream &ost);
   void print_trihedral_pairs(std::ostream &ost);
   void latex_half_double_six(std::ostream &ost, int idx);
   void latex_table_of_Eckardt_points(std::ostream &ost);
   void latex_table_of_tritangent_planes(std::ostream &ost);
   void print_line(std::ostream &ost, int rk);
   void print_Schlaefli_labelling(std::ostream &ost);
   void print_set_of_lines_tex(std::ostream &ost, long int *v, int len);
   void latex_table_of_clebsch_maps(std::ostream &ost);
   void print_half_double_sixes_in_GAP();
   int identify_Eckardt_point(
        int line1, int line2, int line3, int verbose_level);

};

// ##############################################################################
// seventytwo_cases.cpp
// ##############################################################################

//! description of a Clebsch map with a fixed tritangent plane

class seventytwo_cases {

public:

   surface_domain *Surf;

   int f;

   int tritangent_plane_idx;  // = t
      // the tritangent plane picked for the Clebsch map,
      // using the Schlaefli labeling, in [0,44].


   int three_lines_idx[3];
```

```
   // the index into Lines[] of the
   // three lines in the chosen tritangent plane
   // This is computed from the Schlaefli labeling
   // using the eckardt point class.

long int three_lines[3];
   // the three lines in the chosen tritangent plane


long int tritangent_plane_rk;

int Basis_pi[16];
int Basis_pi_inv[17]; // in case it is semilinear

int line_idx;  // = i
   // the index of the line chosen to be P1,P2 in three_lines[3]
   // three_lines refers to class surfaces_arc_lifting_upstep

int m1, m2, m3;
   // rearrangement of three_lines_idx[3]
   // m1 = line_idx is the line through P1 and P2.
   // m2 and m3 are the two other lines.

int l1, l2;
   // the indices of the two lines defining the Clebsch map.
   // They pass through m1.

int line_l1_l2_idx; // = j

int transversals[5];
   // the 5 transversals of l1 and l2 in Schlaefli labeling

long int transversals4[4];
   // the 4 transversals different from m1 in Schlaefli labeling

long int half_double_six[6];
   // long int because surf->find_half_double_six() requires it that way

int half_double_six_index;

long int P6[6];
   // the points of intersection of l1, l2, and of the 4 transversals
   // with the tritangent plane

long int P6a[6];
   // the arc after the plane has been moved
```

```
long int L1, L2; // images of l1 and l2 under Alpha1 * Alpha2 * Beta1 * Beta2

long int P6_local[6];
   // the moved arc in local coordinates

long int P6_local_canonical[6];
   // the canonical form of P6_local[]

long int P6_perm[6];
long int P6_perm_mapped[6];
long int pair[2];
int the_rest[4];

int orbit_not_on_conic_idx;
int pair_orbit_idx;

int partition_orbit_idx;
int the_partition4[4];

int f2;

seventytwo_cases();
~seventytwo_cases();
void init(
      surface_domain *Surf,
      int f, int tritangent_plane_idx,
      int *three_lines_idx, long int *three_lines,
      int line_idx, int m1, int m2, int m3,
      int line_l1_l2_idx, int l1, int l2);
void compute_arc(
      surface_object *SO, int verbose_level);
// We have chosen a tritangent planes
// and we know the three lines m1, m2, m3 in it.
// The lines l1 and l2 intersect m1 in the first two points.
// Computes the 5 transversals to the two lines l1 and l2.
// One of these lines must be m1, so we remove that to have 4 lines.
// These 4 lines intersect the two other lines m2 and m3 in the other 4 points.

// This makes up the arc of 6 points.
// They will be stored in P6[6].
void compute_partition(int verbose_level);
void compute_half_double_six(
      surface_object *SO, int verbose_level);
void print();
void report_seventytwo_maps_line(std::ostream &ost);
void report_seventytwo_maps_top(std::ostream &ost);
void report_seventytwo_maps_bottom(
```

```
        std::ostream &ost);
   void report_single_Clebsch_map(
        std::ostream &ost, int verbose_level);
   void report_Clebsch_map_details(
        std::ostream &ost, surface_object *SO, int verbose_level);
   void report_Clebsch_map_aut_coset(
        std::ostream &ost, int coset,
        int relative_order, int verbose_level);
};


// ##############################################################################
// smooth_surface_object_properties.cpp
// ##############################################################################

//! smooth cubic surfaces in PG(3,q) have 27 lines


class smooth_surface_object_properties {

public:

   surface_object *SO;


   long int *Tritangent_plane_rk; // [45]
      // list of tritangent planes in Schlaefli labeling
   int nb_tritangent_planes;

   long int *Lines_in_tritangent_planes; // [nb_tritangent_planes * 3]

   long int *Trihedral_pairs_as_tritangent_planes; // [nb_trihedral_pairs * 6]


   long int *All_Planes; // [nb_trihedral_pairs * 6]
   int *Dual_point_ranks; // [nb_trihedral_pairs * 6]



   smooth_surface_object_properties();
   ~smooth_surface_object_properties();
   void init(surface_object *SO, int verbose_level);
   void compute_tritangent_planes_by_rank(int verbose_level);
   void compute_Lines_in_tritangent_planes(int verbose_level);
   void compute_Trihedral_pairs_as_tritangent_planes(int verbose_level);
   void compute_planes_and_dual_point_ranks(int verbose_level);
   void print_planes_in_trihedral_pairs(std::ostream &ost);
```

```
   void print_tritangent_planes(std::ostream &ost);
   void print_single_tritangent_plane(
         std::ostream &ost, int plane_idx);

   void latex_table_of_trihedral_pairs(
         std::ostream &ost, int *T, int nb_T);
   void latex_trihedral_pair(std::ostream &ost, int t_idx);
   void make_equation_in_trihedral_form(int t_idx,
      int *F_planes, int *G_planes, int &lambda, int *equation,
      int verbose_level);
   void print_equation_in_trihedral_form(
         std::ostream &ost,
      int *F_planes, int *G_planes, int lambda);
   void print_equation_in_trihedral_form_equation_only(
         std::ostream &ost,
      int *F_planes, int *G_planes, int lambda);
   void make_and_print_equation_in_trihedral_form(
         std::ostream &ost, int t_idx);
   void latex_table_of_trihedral_pairs_and_clebsch_system(
      std::ostream &ost, int *T, int nb_T);
   void latex_trihedral_pair(
         std::ostream &ost, int *T, long int *TE);
   void latex_table_of_trihedral_pairs(std::ostream &ost);
   void print_Steiner_and_Eckardt(std::ostream &ost);

};



// ##############################################################################
// surface_domain.cpp
// ##############################################################################

//! cubic surfaces in PG(3,q)


class surface_domain {

public:
   int q;
   int n; // = 4
   int n2; // = 2 * n

   field_theory::finite_field *F;

   geometry::projective_space *P; // PG(3,q)
   geometry::projective_space *P2; // PG(2,q)
```

```
geometry::grassmann *Gr; // Gr_{4,2}
geometry::grassmann *Gr3; // Gr_{4,3}

long int nb_lines_PG_3;
int nb_pts_on_surface_with_27_lines; // q^2 + 7q + 1

orthogonal_geometry::orthogonal *O;
geometry::klein_correspondence *Klein;


int Basis0[16];
int Basis1[16];
int Basis2[16];

// used in line_to_wedge and klein_to_wedge:
int *v; // [n]
int *v2; // [(n * (n-1)) / 2]
int *w2; // [(n * (n-1)) / 2]


schlaefli *Schlaefli;

surface_polynomial_domains *PolynomialDomains;

surface_domain();
~surface_domain();
void init(
      field_theory::finite_field *F,
      int verbose_level);
void unrank_point(int *v, long int rk);
long int rank_point(int *v);
void unrank_plane(int *v, long int rk);
long int rank_plane(int *v);
void enumerate_points(int *coeff,
      std::vector<long int> &Pts,
      int verbose_level);
void substitute_semilinear(int *coeff_in, int *coeff_out,
   int f_semilinear, int frob, int *Mtx_inv, int verbose_level);
void list_starter_configurations(
      long int *Lines, int nb_lines,
      data_structures::set_of_sets *line_intersections,
      int *&Table, int &N,
   int verbose_level);
void create_starter_configuration(int line_idx, int subset_idx,
      data_structures::set_of_sets *line_neighbors,
      long int *Lines, long int *S,
```

```
   int verbose_level);
void wedge_to_klein(int *W, int *K);
void klein_to_wedge(int *K, int *W);
long int line_to_wedge(long int line_rk);
void line_to_wedge_vec(
      long int *Line_rk, long int *Wedge_rk, int len);
void line_to_klein_vec(
      long int *Line_rk, long int *Klein_rk, int len);
long int klein_to_wedge(long int klein_rk);
void klein_to_wedge_vec(
      long int *Klein_rk, long int *Wedge_rk, int len);
void save_lines_in_three_kinds(std::string &fname_csv,
   long int *Lines_wedge, long int *Lines,
   long int *Lines_klein, int nb_lines);
int build_surface_from_double_six_and_count_Eckardt_points(
      long int *double_six, int verbose_level);


// surface_domain2.cpp:
void create_equations_for_pencil_of_surfaces_from_trihedral_pair(
   int *The_six_plane_equations, int *The_surface_equations,
   int verbose_level);
   // The_surface_equations[(q + 1) * 20]
long int plane_from_three_lines(
      long int *three_lines, int verbose_level);
void Trihedral_pairs_to_planes(
      long int *Lines, long int *Planes_by_rank,
   int verbose_level);
   // Planes_by_rank[nb_trihedral_pairs * 6]
void prepare_system_from_FG(
      int *F_planes, int *G_planes,
   int lambda, int *&system, int verbose_level);
void compute_nine_lines(int *F_planes, int *G_planes,
   long int *nine_lines, int verbose_level);
void compute_nine_lines_by_dual_point_ranks(
      long int *F_planes_rank,
   long int *G_planes_rank,
   long int *nine_lines, int verbose_level);
void tritangent_plane_to_trihedral_pair_and_position(
   int tritangent_plane_idx,
   int &trihedral_pair_idx,
   int &position, int verbose_level);
void do_arc_lifting_with_two_lines(
   long int *Arc6, int p1_idx, int p2_idx, int partition_rk,
   long int line1, long int line2,
   int *coeff20, long int *lines27,
   int verbose_level);
```

```
void compute_local_coordinates_of_arc(
      long int *P6, long int *P6_local, int verbose_level);




// surface_domain_lines.cpp:
void init_Schlaefli(int verbose_level);
void unrank_line(int *v, long int rk);
void unrank_lines(int *v, long int *Rk, int nb);
long int rank_line(int *v);
void build_cubic_surface_from_lines(
      int len, long int *S, int *coeff,
   int verbose_level);
int rank_of_system(int len, long int *S,
      int verbose_level);
void create_system(int len, long int *S,
      int *&System, int &nb_rows, int verbose_level);
void compute_intersection_points(int *Adj,
   long int *Lines, int nb_lines,
   long int *&Intersection_pt,
   int verbose_level);
void compute_intersection_points_and_indices(int *Adj,
   long int *Points, int nb_points,
   long int *Lines, int nb_lines,
   int *&Intersection_pt, int *&Intersection_pt_idx,
   int verbose_level);
void lines_meet3_and_skew3(
      long int *lines_meet3, long int *lines_skew3,
   long int *&lines, int &nb_lines, int verbose_level);
void perp_of_three_lines(
      long int *three_lines, long int *&perp, int &perp_sz,
   int verbose_level);
int perp_of_four_lines(
      long int *four_lines, long int *trans12, int &perp_sz,
   int verbose_level);
int rank_of_four_lines_on_Klein_quadric(long int *four_lines,
   int verbose_level);
int create_double_six_from_five_lines_with_a_common_transversal(
   long int *five_pts, long int *double_six,
   int verbose_level);
int create_double_six_from_six_disjoint_lines(long int *single_six,
      long int *double_six, int verbose_level);
void create_the_fifteen_other_lines(long int *double_six,
   long int *fifteen_other_lines, int verbose_level);
int test_double_six_property(long int *S12, int verbose_level);
void compute_adjacency_matrix_of_line_intersection_graph(
```

```
   int *&Adj,
   long int *S, int n, int verbose_level);
void compute_adjacency_matrix_of_line_disjointness_graph(
   int *&Adj,
   long int *S, int n, int verbose_level);
void compute_points_on_lines(
      long int *Pts_on_surface,
      int nb_points_on_surface,
      long int *Lines, int nb_lines,
      data_structures::set_of_sets *&pts_on_lines,
      int *&f_is_on_line,
      int verbose_level);
int compute_rank_of_any_four(
      long int *&Rk, int &nb_subsets, long int *lines,
   int sz, int verbose_level);
void rearrange_lines_according_to_a_given_double_six(
      long int *Lines,
      int *given_double_six,
      long int *New_lines,
      int verbose_level);
void rearrange_lines_according_to_double_six(long int *Lines,
   int verbose_level);
void rearrange_lines_according_to_starter_configuration(
   long int *Lines, long int *New_lines,
   int line_idx, int subset_idx, int *Adj,
   data_structures::set_of_sets *line_intersections,
   int verbose_level);
int intersection_of_four_lines_but_not_b6(int *Adj,
   int *four_lines_idx, int b6, int verbose_level);
int intersection_of_five_lines(
      int *Adj, int *five_lines_idx,
   int verbose_level);
void rearrange_lines_according_to_a_given_double_six(
      long int *Lines,
   long int *New_lines, long int *double_six, int verbose_level);
void create_lines_from_plane_equations(
      int *The_plane_equations,
   long int *Lines, int verbose_level);
void create_remaining_fifteen_lines(
   long int *double_six, long int *fifteen_lines,
   int verbose_level);
long int compute_cij(long int *double_six,
   int i, int j, int verbose_level);
int compute_transversals_of_any_four(
      long int *&Trans, int &nb_subsets,
      long int *lines, int sz, int verbose_level);
```

```
// surface_domain_io.cpp:
void print_equation(std::ostream &ost, int *coeffs);
void print_equation_maple(std::stringstream &ost, int *coeffs);
void print_equation_tex(std::ostream &ost, int *coeffs);
void print_equation_with_line_breaks_tex(
      std::ostream &ost, int *coeffs);
void print_equation_tex_lint(std::ostream &ost, long int *coeffs);
void latex_double_six(std::ostream &ost, long int *double_six);
void make_spreadsheet_of_lines_in_three_kinds(
      data_structures::spreadsheet *&Sp,
   long int *Wedge_rk, long int *Line_rk,
   long int *Klein_rk, int nb_lines,
   int verbose_level);
void print_equation_wrapped(
      std::ostream &ost, int *the_equation);
void print_lines_tex(
      std::ostream &ost, long int *Lines, int nb_lines);
void print_one_line_tex(
      std::ostream &ost,
      long int *Lines, int nb_lines, int idx);
void print_trihedral_pair_in_dual_coordinates_in_GAP(
   long int *F_planes_rank, long int *G_planes_rank);
void print_basics(std::ostream &ost);
void sstr_line_label(std::stringstream &sstr, long int pt);
void make_table_of_surfaces(int verbose_level);
void make_table_of_surfaces_detailed(
      int *Q_table, int Q_table_len, int verbose_level);
void make_table_of_surfaces2(
      std::ostream &ost,
      int *Q_table, int Q_table_len, int verbose_level);
void table_top(std::ostream &ost);
void table_bottom(std::ostream &ost);
void compute_table_E(
      int *field_orders, int nb_fields,
      long int *&Table,
      int *&Q, int &nb_Q,
      int *&E, int &nb_E_types, int verbose_level);


// surface_domain_families.cpp:
void create_equation_general_abcd(int a, int b, int c, int d,
      int *coeff, int verbose_level);
void create_equation_Cayley_klmn(int k, int l, int m, int n,
      int *coeff, int verbose_level);
void create_equation_bes(int a, int c, int *coeff, int verbose_level);
void create_equation_F13(int a, int *coeff, int verbose_level);
void create_equation_G13(int a, int *coeff, int verbose_level);
```

```
   surface_object *create_surface_general_abcd(int a, int b, int c, int d,
      int verbose_level);
   surface_object *create_surface_bes(int a, int c,
      int verbose_level);
   surface_object *create_surface_F13(int a, int verbose_level);
   surface_object *create_surface_G13(int a, int verbose_level);
   surface_object *create_Eckardt_surface(int a, int b,
      int &alpha, int &beta,
      int verbose_level);
   void create_equation_Eckardt_surface(
         int a, int b, int *coeff, int verbose_level);
   int test_Eckardt_form_alpha_beta(int *coeff, int &alpha, int &beta,
      int verbose_level);
   void create_Eckardt_double_six(long int *double_six, int a, int b,
      int verbose_level);
   void create_Eckardt_fifteen_lines(
         long int *fifteen_lines, int a, int b,
      int verbose_level);

};

void callback_surface_domain_sstr_line_label(
      std::stringstream &sstr, long int pt, void *data);

// ##############################################################################
// surface_object_properties.cpp
// ##############################################################################

//! properties of a particular cubic surface in PG(3,q), as defined by an object
of class surface_object


class surface_object_properties {

public:

   surface_object *SO;


   // point properties:

   data_structures::set_of_sets *pts_on_lines;
      // points are stored as indices into Pts[]
   int *f_is_on_line; // [SO->nb_pts]
```

```
long int *Eckardt_points;
   // the orbiter ranks of the Eckardt points
int *Eckardt_points_index;
   // index into SO->Pts
int *Eckardt_points_schlaefli_labels;
   // Schlaefli labels
int *Eckardt_point_bitvector_in_Schlaefli_labeling;
   // true if the i-th Eckardt point
   // in the Schlaefli labeling is present
int nb_Eckardt_points;

int *Eckardt_points_line_type;
   // [nb_Eckardt_points + 1]
int *Eckardt_points_plane_type;
   // [SO->Surf->P->Nb_subspaces[2]]



data_structures::set_of_sets *lines_on_point;
data_structures::tally *Type_pts_on_lines;
data_structures::tally *Type_lines_on_point;


long int *Hesse_planes;
int nb_Hesse_planes;
int *Eckardt_point_Hesse_plane_incidence;
   // [nb_Eckardt_points * nb_Hesse_planes]


int nb_axes;
int *Axes_index;
   // [nb_axes] two times the index
   // into trihedral pairs +0 or +1
long int *Axes_Eckardt_points;
   // [nb_axes * 3] the Eckardt points
   // in Schlaefli labels that lie on the axes
long int *Axes_line_rank;


long int *Double_points;
int *Double_points_index;
int nb_Double_points;

long int *Single_points;
int *Single_points_index;
int nb_Single_points;
```

```
long int *Pts_not_on_lines;
int nb_pts_not_on_lines;

int nb_planes;
int *plane_type_by_points; // [nb_planes]
int *plane_type_by_lines; // [nb_planes]
data_structures::tally *C_plane_type_by_points;


// only for surfaces with 27 lines:

smooth_surface_object_properties *SmoothProperties;


int *Adj_line_intersection_graph;
   // [SO->nb_lines * SO->nb_lines]
data_structures::set_of_sets *Line_neighbors;
int *Line_intersection_pt;
   // [SO->nb_lines * SO->nb_lines]
int *Line_intersection_pt_idx;
   // [SO->nb_lines * SO->nb_lines]


int *gradient;
   // [4 * SO->Surf->Poly2_4->get_nb_monomials()]

long int *singular_pts;
int nb_singular_pts;
int nb_non_singular_pts;

long int *tangent_plane_rank_global;
   // [SO->nb_pts]
long int *tangent_plane_rank_dual;
   // [nb_non_singular_pts]

surface_object_properties();
~surface_object_properties();
void init(surface_object *SO, int verbose_level);
void compute_properties(int verbose_level);
void compute_axes(int verbose_level);
void compute_gradient(int verbose_level);
void compute_singular_points_and_tangent_planes(int verbose_level);
void compute_adjacency_matrix_of_line_intersection_graph(
   int verbose_level);
int Adj_ij(int i, int j);
void compute_plane_type_by_points(int verbose_level);
```

```
void report_properties(std::ostream &ost, int verbose_level);
void report_properties_simple(std::ostream &ost, int verbose_level);
void print_line_intersection_graph(std::ostream &ost);
void print_adjacency_list(std::ostream &ost);
void print_adjacency_matrix(std::ostream &ost);
void print_adjacency_matrix_with_intersection_points(
      std::ostream &ost);
void print_neighbor_sets(std::ostream &ost);


void print_plane_type_by_points(std::ostream &ost);
void print_lines(std::ostream &ost);
void print_lines_with_points_on_them(std::ostream &ost);
void print_equation(std::ostream &ost);
void print_summary(std::ostream &ost);
void print_affine_points_in_source_code(std::ostream &ost);
void print_points(std::ostream &ost);
void print_Eckardt_points(std::ostream &ost);
void print_Hesse_planes(std::ostream &ost);
void print_axes(std::ostream &ost);
void print_singular_points(std::ostream &ost);
void print_double_points(std::ostream &ost);
void print_single_points(std::ostream &ost);
void print_points_on_surface(std::ostream &ost);
void print_all_points_on_surface(std::ostream &ost);
void print_points_on_lines(std::ostream &ost);
void print_points_on_surface_but_not_on_a_line(
      std::ostream &ost);
void print_double_sixes(std::ostream &ost);
void print_half_double_sixes(std::ostream &ost);
void print_half_double_sixes_numerically(std::ostream &ost);
void print_trihedral_pairs(std::ostream &ost);
void print_trihedral_pairs_numerically(std::ostream &ost);

int compute_transversal_line(int line_a, int line_b,
   int verbose_level);
void compute_transversal_lines(
   int line_a, int line_b, int *transversals5,
   int verbose_level);
void clebsch_map_find_arc_and_lines(long int *Clebsch_map,
   long int *Arc, long int *Blown_up_lines, int verbose_level);
void clebsch_map_latex(std::ostream &ost,
      long int *Clebsch_map, int *Clebsch_coeff);
void compute_reduced_set_of_points_not_on_lines_wrt_P(
      int P_idx,
      int *&f_deleted, int verbose_level);
int test_full_del_pezzo(
```

```
         int P_idx, int *f_deleted, int verbose_level);
   void create_summary_file(std::string &fname,
         std::string &surface_label,
         std::string &col_postfix,
         int verbose_level);

};



// ##############################################################################
// surface_object.cpp
// ##############################################################################

//! a particular cubic surface in PG(3,q), given by its equation


class surface_object {

public:
   int q;
   field_theory::finite_field *F;
   surface_domain *Surf;

   long int *Pts; // in increasing order
   int nb_pts;


   long int *Lines;
   int nb_lines;

   int eqn[20];

   surface_object_properties *SOP;



   surface_object();
   ~surface_object();
   void init_equation_points_and_lines_only(
         surface_domain *Surf, int *eqn,
      int verbose_level);
   void init_equation(
         surface_domain *Surf, int *eqn, int verbose_level);
   void enumerate_points(int verbose_level);
   void enumerate_points_and_lines(int verbose_level);
   void find_real_lines(
```

```
        std::vector<long int> &The_Lines,
        int verbose_level);
   void init_with_27_lines(
        surface_domain *Surf, long int *Lines27, int *eqn,
      int f_find_double_six_and_rearrange_lines,
      int verbose_level);
   void compute_properties(int verbose_level);
   void recompute_properties(int verbose_level);
   void find_double_six_and_rearrange_lines(
        long int *Lines, int verbose_level);
   void identify_lines(
        long int *lines, int nb_lines, int *line_idx,
      int verbose_level);
   void print_nine_lines_latex(
        std::ostream &ost, long int *nine_lines,
      int *nine_lines_idx);
   int find_point(long int P, int &idx);
   void export_something(
        std::string &what,
        std::string &fname_base, int verbose_level);
   void latex_double_six(std::ostream &ost, int idx);
   void latex_double_six_wedge(
        std::ostream &ost, int idx);
   void latex_double_six_Klein(
        std::ostream &ost, int idx);
   void latex_double_six_Pluecker_coordinates_transposed(
        std::ostream &ost, int idx);
   void latex_double_six_Klein_transposed(
        std::ostream &ost, int idx);
   void print_lines_tex(std::ostream &ost);
   void print_one_line_tex(std::ostream &ost, int idx);


};




// ##############################################################################
// surface_polynomial_domains.cpp
// ##############################################################################

//! polynomial domains associated with cubic surfaces


class surface_polynomial_domains {
```

```
public:

   surface_domain *Surf;

   int nb_monomials; // = 20


   ring_theory::homogeneous_polynomial_domain *Poly1;
      // linear polynomials in three variables
   ring_theory::homogeneous_polynomial_domain *Poly2;
      // quadratic polynomials in three variables
   ring_theory::homogeneous_polynomial_domain *Poly3;
      // cubic polynomials in three variables

   ring_theory::homogeneous_polynomial_domain *Poly1_x123;
      // linear polynomials in three variables
   ring_theory::homogeneous_polynomial_domain *Poly2_x123;
      // quadratic polynomials in three variables
   ring_theory::homogeneous_polynomial_domain *Poly3_x123;
      // cubic polynomials in three variables
   ring_theory::homogeneous_polynomial_domain *Poly4_x123;
      // quartic polynomials in three variables

   ring_theory::homogeneous_polynomial_domain *Poly1_4;
      // linear polynomials in four variables
   ring_theory::homogeneous_polynomial_domain *Poly2_4;
      // quadratic polynomials in four variables
   ring_theory::homogeneous_polynomial_domain *Poly3_4;
      // cubic polynomials in four variables

   ring_theory::partial_derivative *Partials; // [4]


   int f_has_large_polynomial_domains;
   ring_theory::homogeneous_polynomial_domain *Poly2_27;
   ring_theory::homogeneous_polynomial_domain *Poly4_27;
   ring_theory::homogeneous_polynomial_domain *Poly6_27;
   ring_theory::homogeneous_polynomial_domain *Poly3_24;

   int nb_monomials2, nb_monomials4, nb_monomials6;
   int nb_monomials3;

   int *Clebsch_Pij;
   int **Clebsch_P;
   int **Clebsch_P3;

   int *Clebsch_coeffs; // [4 * Poly3->nb_monomials * nb_monomials3]
```

```
int **CC; // [4 * Poly3->nb_monomials]



surface_polynomial_domains();
~surface_polynomial_domains();
void init(surface_domain *Surf, int verbose_level);
void init_large_polynomial_domains(int verbose_level);
void label_variables_3(
      ring_theory::homogeneous_polynomial_domain *HPD,
   int verbose_level);
void label_variables_x123(
      ring_theory::homogeneous_polynomial_domain *HPD,
   int verbose_level);
void label_variables_4(
      ring_theory::homogeneous_polynomial_domain *HPD,
   int verbose_level);
void label_variables_27(
      ring_theory::homogeneous_polynomial_domain *HPD,
   int verbose_level);
void label_variables_24(
      ring_theory::homogeneous_polynomial_domain *HPD,
   int verbose_level);
int index_of_monomial(int *v);
void multiply_conic_times_linear(
      int *six_coeff, int *three_coeff,
   int *ten_coeff, int verbose_level);
void multiply_linear_times_linear_times_linear(
      int *three_coeff1,
   int *three_coeff2, int *three_coeff3, int *ten_coeff,
   int verbose_level);
void multiply_linear_times_linear_times_linear_in_space(
   int *four_coeff1, int *four_coeff2, int *four_coeff3,
   int *twenty_coeff, int verbose_level);
void multiply_Poly2_3_times_Poly2_3(int *input1, int *input2,
   int *result, int verbose_level);
void multiply_Poly1_3_times_Poly3_3(int *input1, int *input2,
   int *result, int verbose_level);
void clebsch_cubics(int verbose_level);
void multiply_222_27_and_add(int *M1, int *M2, int *M3,
   int scalar, int *MM, int verbose_level);
void minor22(int **P3, int i1, int i2, int j1, int j2,
   int scalar, int *Ad, int verbose_level);
void multiply42_and_add(int *M1, int *M2, int *MM,
   int verbose_level);
void split_nice_equation(int *nice_equation, int *&f1,
   int *&f2, int *&f3, int verbose_level);
```

```
   void assemble_tangent_quadric(int *f1, int *f2, int *f3,
      int *&tangent_quadric, int verbose_level);
   void compute_gradient(
         int *equation20, int *&gradient, int verbose_level);
   long int compute_tangent_plane(
         int *pt_coords, int *equation20, int verbose_level);
   void print_clebsch_P(std::ostream &ost);
   void print_clebsch_P_matrix_only(std::ostream &ost);
   void print_clebsch_cubics(std::ostream &ost);
   void print_system(std::ostream &ost, int *system);
   void print_polynomial_domains(std::ostream &ost);
   void print_equation_in_trihedral_form(std::ostream &ost,
      int *the_six_plane_equations,
      int lambda, int *the_equation);

};



// ##############################################################################
// web_of_cubic_curves.cpp
// ##############################################################################

//! a web of cubic curves which is used to create an algebraic variety


class web_of_cubic_curves {

public:
   surface_domain *Surf;


   long int arc6[6];

   eckardt_point_info *E;

   int *E_idx;

   int *T_idx;
   int nb_T;


   int base_curves4[4];
   int t_idx0;
   long int row_col_Eckardt_points[6];
   int six_curves[6 * 10];
   int *Web_of_cubic_curves; // [45 * 10]
```

```
int *Tritangent_plane_equations; // [45 * 4]
int *base_curves; // [4 * 10]
long int *The_plane_rank; // [45]
long int *The_plane_duals; // [45]
long int *Dual_point_ranks; // [nb_T * 6]
long int Lines27[27];

web_of_cubic_curves();
~web_of_cubic_curves();
void init(surface_domain *Surf,
      long int *arc6, int verbose_level);
void compute_web_of_cubic_curves(
      long int *arc6, int verbose_level);
void rank_of_foursubsets(int *&rk, int &N, int verbose_level);
void create_web_and_equations_based_on_four_tritangent_planes(
      long int *arc6, int *base_curves4,
      int verbose_level);
void find_Eckardt_points(int verbose_level);
void find_trihedral_pairs(int verbose_level);
void extract_six_curves_from_web(
   int verbose_level);
void create_surface_equation_from_trihedral_pair(
      long int *arc6,
   int t_idx, int *surface_equation,
   int &lambda,
   int verbose_level);
void create_lambda_from_trihedral_pair_and_arc(
   long int *arc6, int t_idx,
   int &lambda, int &lambda_rk,
   int verbose_level);
void find_point_not_on_six_curves(
      int &pt, int &f_point_was_found,
   int verbose_level);
void print_lines(std::ostream &ost);
void print_trihedral_plane_equations(std::ostream &ost);
void print_the_six_plane_equations(
   int *The_six_plane_equations,
   long int *plane6, std::ostream &ost);
void print_surface_equations_on_line(
   int *The_surface_equations,
   int lambda, int lambda_rk, std::ostream &ost);
void print_dual_point_ranks(std::ostream &ost);
void print_Eckardt_point_data(
      std::ostream &ost, int verbose_level);
void report_basics(std::ostream &ost, int verbose_level);
void report(std::ostream &ost, int verbose_level);
void print_web_of_cubic_curves(long int *arc6, std::ostream &ost);
```

```
};


}}}


#endif /* SRC_LIB_FOUNDATIONS_ALGEBRAIC_GEOMETRY_ALGEBRAIC_GEOMETRY_H_ */
```

3.4 Coding Theory

```
// coding_theory.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005


#ifndef ORBITER_SRC_LIB_FOUNDATIONS_CODING_THEORY_CODING_THEORY_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_CODING_THEORY_CODING_THEORY_H_


namespace orbiter {
namespace layer1_foundations {
namespace coding_theory {


// ##############################################################################
// code_diagram.cpp:
// ##############################################################################

//! diagram of a code in Hamming space


class code_diagram {
public:

   long int *Words;
   int nb_words;

   std::string label;

   int n;
   long int N;

   int nb_rows, nb_cols;
   int *v;
   int *Index_of_codeword;
   int *Place_values;
   int *Characteristic_function;
   int *Distance;
   int *Distance_H;
      // the distance of a word to the code.
      // Can be used to detect deep holes.
```

```
   code_diagram();
   ~code_diagram();

   void init(std::string &label,
         long int *Words, int nb_words,
         int n, int verbose_level);
   void place_codewords(int verbose_level);
   void place_metric_balls(
         int radius_of_metric_ball, int verbose_level);
   void compute_distances(int verbose_level);
   void dimensions(int n, int &nb_rows, int &nb_cols);
   void place_binary(long int h, int &i, int &j);
   void place_binary(int *v, int n, int &i, int &j);
   void convert_to_binary(int n, long int h, int *v);
   void print_binary(int n, int *v);
   void save_distance(int verbose_level);
   void save_distance_H(int verbose_level);
   void save_diagram(int verbose_level);
   void save_char_func(int verbose_level);
   void report(int verbose_level);

};


// ##############################################################################
// coding_theory_domain.cpp:
// ##############################################################################

//! various functions related to coding theory


class coding_theory_domain {
public:

   coding_theory_domain();
   ~coding_theory_domain();



   void make_mac_williams_equations(
         ring_theory::longinteger_object *&M,
         int n, int k, int q, int verbose_level);
   void make_table_of_bounds(
         int n_max, int q, int verbose_level);
   void make_gilbert_varshamov_code(
         int n, int k, int d,
```

```
      field_theory::finite_field *F,
      int *&genma, int *&checkma,
      int verbose_level);
void make_gilbert_varshamov_code_recursion(
      field_theory::finite_field *F,
      int n, int k, int d, long int N_points,
      long int *set, int *f_forbidden, int level,
      int verbose_level);

int gilbert_varshamov_lower_bound_for_d(
      int n, int k, int q, int verbose_level);
int singleton_bound_for_d(
      int n, int k, int q, int verbose_level);
int hamming_bound_for_d(
      int n, int k, int q, int verbose_level);
int plotkin_bound_for_d(
      int n, int k, int q, int verbose_level);
int griesmer_bound_for_d(
      int n, int k, int q, int verbose_level);
int griesmer_bound_for_n(
      int k, int d, int q, int verbose_level);

void do_make_macwilliams_system(
      int q, int n, int k, int verbose_level);
void make_Hamming_space_distance_matrix(
      int n, field_theory::finite_field *F,
      int f_projective, int verbose_level);
void compute_and_print_projective_weights(
      std::ostream &ost, field_theory::finite_field *F,
      int *M, int n, int k, int verbose_level);
int code_minimum_distance(
      field_theory::finite_field *F, int n, int k,
   int *code, int verbose_level);
   // code[k * n]
void make_codewords_sorted(
      field_theory::finite_field *F,
      int n, int k,
      int *genma, // [k * n]
      long int *&codewords, // q^k
      long int &N,
      int verbose_level);
void make_codewords(
      field_theory::finite_field *F,
      int n, int k,
      int *genma, // [k * n]
      long int *&codewords, // q^k
      long int &N,
```

```
      int verbose_level);
void codewords_affine(
      field_theory::finite_field *F, int n, int k,
   int *code, // [k * n]
   long int *codewords, // q^k
   int verbose_level);
void codewords_table(field_theory::finite_field *F,
      int n, int k,
   int *code, // [k * n]
   int *&codewords, // [q^k * n]
   long int &N, // q^k
   int verbose_level);
void code_projective_weight_enumerator(
      field_theory::finite_field *F, int n, int k,
   int *code, // [k * n]
   int *weight_enumerator, // [n + 1]
   int verbose_level);
void code_weight_enumerator(
      field_theory::finite_field *F, int n, int k,
   int *code, // [k * n]
   int *weight_enumerator, // [n + 1]
   int verbose_level);
void code_weight_enumerator_fast(
      field_theory::finite_field *F, int n, int k,
   int *code, // [k * n]
   int *weight_enumerator, // [n + 1]
   int verbose_level);
void code_projective_weights(
      field_theory::finite_field *F, int n, int k,
   int *code, // [k * n]
   int *&weights,
      // will be allocated [N]
      // where N = theta_{k-1}
   int verbose_level);
void mac_williams_equations(
      ring_theory::longinteger_object *&M,
      int n, int k, int q);
void determine_weight_enumerator();
void do_weight_enumerator(
      field_theory::finite_field *F,
      int *M, int m, int n,
      int f_normalize_from_the_left,
      int f_normalize_from_the_right,
      int verbose_level);
void do_minimum_distance(
      field_theory::finite_field *F,
      int *M, int m, int n,
```

```
      int verbose_level);
void matrix_from_projective_set(
      field_theory::finite_field *F,
      int n, int k, long int *columns_set_of_size_n,
      int *genma,
      int verbose_level);
void do_linear_code_through_columns_of_generator_matrix(
      field_theory::finite_field *F,
      int n,
      long int *columns_set, int k,
      int *&genma,
      int verbose_level);
void do_polynomial(
      int n,
      int polynomial_degree,
      int polynomial_nb_vars,
      std::string &polynomial_text,
      int verbose_level);
void do_sylvester_hadamard(
      field_theory::finite_field *F3,
      int n,
      int verbose_level);
void field_reduction(
      field_theory::finite_field *FQ,
      field_theory::finite_field *Fq,
      std::string &label,
      int m, int n, std::string &genma_text,
      int verbose_level);
// creates a field_theory::subfield_structure object
void field_induction(std::string &fname_in,
      std::string &fname_out, int nb_bits,
      int verbose_level);
void encode_text_5bits(std::string &text,
      std::string &fname, int verbose_level);
int Hamming_distance(int *v1, int *v2, int n);
int Hamming_distance_binary(int a, int b, int n);
void fixed_code(
      field_theory::finite_field *F,
      int n, int k, int *genma,
      long int *perm,
      int *&subcode_genma, int &subcode_k,
      int verbose_level);
void polynomial_representation_of_boolean_function(
      field_theory::finite_field *F,
      std::string &label_txt,
      long int *Words,
      int nb_words, int n,
```

```
         int verbose_level);
   // creates a combinatorics::boolean_function_domain object

   // mindist.cpp:
   int mindist(int n, int k, int q, int *G,
      int f_verbose_level, int idx_zero, int idx_one,
      int *add_table, int *mult_table);
   //Main routine for the code minimum distance computation.
   //The tables are only needed if $q = p^f$ with $f > 1$.
   //In the GF(p) case, just pass a NULL pointer.

};

// ##############################################################################
// crc_codes.cpp:
// ##############################################################################

//! algorithms for CRC codes


class crc_codes {
public:

   crc_codes();
   ~crc_codes();

   // crc_codes_search.cpp:
   void find_CRC_polynomials(field_theory::finite_field *F,
         int t, int da, int dc,
         int verbose_level);
   void search_for_CRC_polynomials(int t,
         int da, int *A, int dc, int *C,
         int i, field_theory::finite_field *F,
         long int &nb_sol,
         std::vector<std::vector<int> > &Solutions,
         int verbose_level);
   void search_for_CRC_polynomials_binary(int t,
         int da, int *A, int dc, int *C, int i,
         long int &nb_sol,
         std::vector<std::vector<int> > &Solutions,
         int verbose_level);
   int test_all_two_bit_patterns(
         int da, int *A, int dc, int *C,
         field_theory::finite_field *F,
         int verbose_level);
   int test_all_three_bit_patterns(
         int da, int *A, int dc, int *C,
```

```
        field_theory::finite_field *F,
        int verbose_level);
int test_all_two_bit_patterns_binary(
        int da, int *A, int dc, int *C,
        int verbose_level);
int test_all_three_bit_patterns_binary(
        int da, int *A, int dc, int *C,
        int verbose_level);
int remainder_is_nonzero(
        int da, int *A, int db, int *B,
        field_theory::finite_field *F);
int remainder_is_nonzero_binary(
        int da, int *A, int db, int *B);

// crc_codes.cpp:
uint16_t crc16(const uint8_t *data, size_t size);
uint32_t crc32(const uint8_t *s, size_t n);
void crc32_test(int block_length, int verbose_level);
void crc256_test_k_subsets(
        int message_length, int R, int k, int verbose_level);
void crc32_remainders(
        int message_length, int verbose_level);
void crc32_remainders_compute(
        int message_length, int R,
        uint32_t *&Crc, int verbose_level);
void introduce_errors(
        crc_options_description *Crc_options_description,
        int verbose_level);
void crc_encode_file_based(
        std::string &fname_in,
        std::string &fname_out,
        std::string &crc_type,
        int block_length, int verbose_level);
void crc_general_file_based(
        std::string &fname_in, std::string &fname_out,
        CRC_type type,
        int block_length, int verbose_level);
enum CRC_type detect_type_of_CRC(std::string &crc_type, int verbose_level);
int get_check_size_in_bytes(enum CRC_type type);
#if 0
void crc16_file_based(
        std::string &fname_in,
        std::string &fname_out,
        int block_length, int verbose_level);
void crc32_file_based(
        std::string &fname_in,
        std::string &fname_out,
```

```
         int block_length, int verbose_level);
   void crc771_file_based(
         std::string &fname_in,
         std::string &fname_out,
         int verbose_level);
#endif
   void check_errors(
         crc_options_description *Crc_options_description,
         int verbose_level);
   void extract_block(
         crc_options_description *Crc_options_description,
         int verbose_level);
   uint16_t NetIpChecksum(uint16_t const *ipHeader, int nWords);
   void CRC_encode_text(
         field_theory::nth_roots *Nth,
         ring_theory::unipoly_object &CRC_poly,
      std::string &text, std::string &fname,
      int verbose_level);

};

// ##############################################################################
// crc_options_description.cpp:
// ##############################################################################

//! options for activities involving CRC codes


class crc_options_description {
public:

   int f_input;
   std::string input_fname;

   int f_output;
   std::string output_fname;

   int f_crc_type;
   std::string crc_type;

   int f_block_length;
   int block_length;

   int f_block_based_error_generator;

   int f_file_based_error_generator;
   int file_based_error_generator_threshold;
```

```
   int f_nb_repeats;
   int nb_repeats;

   int f_threshold;
   int threshold;

   int f_error_log;
   std::string error_log_fname;

   int f_selected_block;
   int selected_block;


   crc_options_description();
   ~crc_options_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// create_BCH_code.cpp:
// ##############################################################################

//! to create a BCH code


class create_BCH_code {
public:

   int n;
   int d;
   field_theory::finite_field *F;

   field_theory::nth_roots *Nth;

   ring_theory::unipoly_object *P;

   int *Selection; // [Nth->Cyc->S->nb_sets]
   int *Sel;  // [nb_sel]
   int nb_sel;

   int degree;
```

```
   int k;
   int *Genma; // [k * n]
   int *generator_polynomial; // [degree + 1]



   create_BCH_code();
   ~create_BCH_code();
   void init(field_theory::finite_field *F,
         int n, int d, int verbose_level);
   void do_report(int verbose_level);
   void report(std::ostream &ost, int verbose_level);


};

// ##############################################################################
// create_RS_code.cpp:
// ##############################################################################

//! to create a RS code


class create_RS_code {
public:

   int n;
   int d;
   field_theory::finite_field *F;

   ring_theory::unipoly_domain *FX; // polynomial ring over F

   ring_theory::unipoly_object *P;

   int degree;
   int k;
   int *Genma; // [k * n]
   int *generator_polynomial; // [degree + 1]



   create_RS_code();
   ~create_RS_code();
   void init(field_theory::finite_field *F,
         int n, int d, int verbose_level);
   void do_report(int verbose_level);
   void report(std::ostream &ost, int verbose_level);
```

```
};


// ##############################################################################
// cyclic_codes.cpp:
// ##############################################################################

//! algorithms for cyclic codes


class cyclic_codes {
public:


   cyclic_codes();
   ~cyclic_codes();

   // cyclic_codes.cpp:
   void make_cyclic_code(int n, int q, int t,
         int *roots, int nb_roots,
         int f_poly, std::string &poly,
         int f_dual,
         std::string &fname_txt,
         std::string &fname_csv,
         int verbose_level);
   // this function creates a finite field,
   // using the given polynomial if necessary
   void generator_matrix_cyclic_code(int n,
         int degree, int *generator_polynomial, int *&M);
   void print_polynomial(
         ring_theory::unipoly_domain &Fq,
         int degree, ring_theory::unipoly_object *coeffs);
   void print_polynomial_tight(
         std::ostream &ost, ring_theory::unipoly_domain &Fq,
         int degree, ring_theory::unipoly_object *coeffs);
   void field_reduction(
         int n, int q, int p, int e, int m,
         field_theory::finite_field &Fp,
         ring_theory::unipoly_domain &Fq,
      int degree, ring_theory::unipoly_object *generator,
      int *&generator_subfield,
      int f_poly, std::string &poly,
      int verbose_level);
   void BCH_generator_polynomial(
         field_theory::finite_field *F,
```

```
         ring_theory::unipoly_object &g, int n,
         int designed_distance, int &bose_distance,
         int &transversal_length, int *&transversal,
         ring_theory::longinteger_object *&rank_of_irreducibles,
         int verbose_level);
   void compute_generator_matrix(
         ring_theory::unipoly_object a, int *&genma,
      int n, int &k, int verbose_level);
   void generator_matrix_cyclic_code(
         field_theory::finite_field *F,
         int n,
         std::string &poly_coeffs,
         int verbose_level);


};


// ##############################################################################
// error_repository.cpp:
// ##############################################################################

//! to remember the errors in a block

class error_repository {

public:

   int nb_errors;
   int allocated_length;
   int * Error_storage;
      // [nb_errors * 2]
      // offset and XOR pattern

   error_repository();
   ~error_repository();
   void init(int verbose_level);
   void add_error(int offset,
         int error_pattern, int verbose_level);
   int search(int offset, int error_pattern,
      int &idx, int verbose_level);

};



// ##############################################################################
```

```
// ttp_codes.cpp:
// ##############################################################################

//! twisted tensor product codes


class ttp_codes {
public:

   ttp_codes();
   ~ttp_codes();

   void twisted_tensor_product_codes(
      field_theory::finite_field *FQ,
      field_theory::finite_field *Fq,
      int f_construction_A, int f_hyperoval,
      int f_construction_B,
      int *&H_subfield, int &m, int &n,
      int verbose_level);
   void create_matrix_M(
      int *&M,
      field_theory::finite_field *FQ,
      field_theory::finite_field *Fq,
      int &m, int &n, int &beta, int &r, int *exponents,
      int f_construction_A, int f_hyperoval, int f_construction_B,
      int f_elements_exponential, std::string &symbol_for_print,
      int verbose_level);
      // int exponents[9]
   void create_matrix_H_subfield(
         field_theory::finite_field *FQ,
         field_theory::finite_field *Fq,
      int *H_subfield, int *C, int *C_inv, int *M, int m, int n,
      int beta, int beta_q,
      int f_elements_exponential,
      std::string &symbol_for_print,
      std::string &symbol_for_print_subfield,
      int f_construction_A, int f_construction_B,
      int verbose_level);
   void tt_field_reduction(
         field_theory::finite_field &F,
         field_theory::finite_field &f,
      int m, int n, int *M, int *MM, int verbose_level);


   void make_tensor_code_9dimensional_as_point_set(
         field_theory::finite_field *F,
      int *&the_set, int &length,
```

```
   int verbose_level);
void make_tensor_code_9_dimensional(int q,
      std::string &override_poly_Q,
      std::string &override_poly,
      int f_hyperoval,
      int *&code, int &length,
      int verbose_level);


void do_tensor(int q,
      std::string &override_poly_Q,
      std::string &override_poly_q,
   int f_construction_A, int f_hyperoval,
   int f_construction_B, int verbose_level);
void action_on_code(
      field_theory::finite_field &F,
      field_theory::finite_field &f, int m, int n,
   int *M, int *H_subfield,
   int *C, int *C_inv, int *A, int *U,
   int *perm, int verbose_level);
void test_cyclic(
      field_theory::finite_field &F,
      field_theory::finite_field &f,
   int *Aut, int *M, int *H_subfield,
   int *C, int *C_inv, int *U,
   int q, int Q, int m, int n,
   int beta, int verbose_level);
void is_cyclic(
      field_theory::finite_field &FQQ,
      field_theory::finite_field &F,
      field_theory::finite_field &f,
   int *Aut, int *M, int *H_subfield,
   int *C, int *C_inv, int *U,
   int q, int Q, int m, int n,
   int beta, int a, int b, int c, int d,
   int verbose_level);
void test_representation(
      field_theory::finite_field &F,
      field_theory::finite_field &f, int Q,
   int beta, int m, int n, int *H_subfield);
void multiply_abcd(
      field_theory::finite_field &F,
   int a1, int b1, int c1, int d1,
   int a2, int b2, int c2, int d2,
   int &a3, int &b3, int &c3, int &d3);
int choose_abcd_first(
      field_theory::finite_field &F,
```

```
        int Q, int &a, int &b, int &c, int &d);
   int choose_abcd_next(
        field_theory::finite_field &F,
        int Q, int &a, int &b, int &c, int &d);
   int choose_abcd_next2(
        field_theory::finite_field &F,
        int Q, int &a, int &b, int &c, int &d);
   void choose_abcd_at_random(
        field_theory::finite_field &F,
        int Q, int &a, int &b, int &c, int &d);
   int compute_mindist(
        field_theory::finite_field &f,
        int m, int n,
        int *generator_matrix, int verbose_level);
   int abcd_term(
        field_theory::finite_field &f,
        int a, int b, int c, int d,
        int e1, int e2, int e3, int e4);
   void do_other_stuff(
        field_theory::finite_field *F,
        field_theory::finite_field *f,
        int beta, int beta_q,
      int *M, int *C, int *C_inv, int *H_subfield,
      int m, int n, int r,
      int f_elements_exponential,
      std::string &symbol_for_print,
      std::string &symbol_for_print_subfield,
      int f_construction_A, int f_hyperoval,
      int f_construction_B,
      int verbose_level);
   void int_submatrix_all_rows(int *A, int m, int n,
      int nb_cols, int *cols, int *B);


};



}}}



#endif /* ORBITER_SRC_LIB_FOUNDATIONS_CODING_THEORY_CODING_THEORY_H_ */
```

3.5 Combinatorics

```
// combinatorics.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005



#ifndef ORBITER_SRC_LIB_FOUNDATIONS_COMBINATORICS_COMBINATORICS_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_COMBINATORICS_COMBINATORICS_H_



namespace orbiter {
namespace layer1_foundations {
namespace combinatorics {


// ##############################################################################
// boolean_function_domain.cpp
// ##############################################################################

//! boolean functions

class boolean_function_domain {

public:
   int n;
   int n2; // n / 2
   int Q; // 2^n
   int bent; // 2^{n/2}
   int near_bent; // 2^{(n+1)/2}
   //int NN;
   ring_theory::longinteger_object *NN; // 2^Q
   int N; // size of PG(n,2)

   field_theory::finite_field *Fq; // the field F2
   //finite_field *FQ; // the field of order 2^n

   ring_theory::homogeneous_polynomial_domain *Poly;
      // Poly[i] = polynomial of degree i in n + 1 variables.
      // i = 1,..,n
   int **A_poly; // [1..n][Poly[i].get_nb_monomials()]
```

```
   int **B_poly; // [1..n][Poly[i].get_nb_monomials()]
   int *Kernel;
   int dim_kernel;


   long int *affine_points; // [Q]
      // affine_points[i] = PG_rank of affine point[i]



   int *v; // [n]
   int *v1; // [n + 1]
   int *w; // [n]
   int *f; // [Q]
   int *f2; // [Q]
   int *F; // [Q]
   int *T; // [Q]
   int *W; // [Q * Q] = Walsh matrix
   int *f_proj;
   int *f_proj2;



   boolean_function_domain();
   ~boolean_function_domain();
   void init(field_theory::finite_field *F2, int n, int verbose_level);
   void setup_polynomial_rings(int verbose_level);
   void compute_polynomial_representation(int *func, int *coeff, int verbose_level
);
   void evaluate_projectively(int *coeff, int *f);
   void evaluate(int *coeff, int *f);
   void raise(int *in, int *out);
   void apply_Walsh_transform(int *in, int *out);
   int is_bent(int *T);
   int is_near_bent(int *T);
};




// ##############################################################################
// brick_domain.cpp
// ##############################################################################

//! a problem of Neil Sloane

class brick_domain {
```

```
public:
   field_theory::finite_field *F;
   int q;
   int nb_bricks;

   brick_domain();
   ~brick_domain();
   void init(field_theory::finite_field *F, int verbose_level);
   void unrank(int rk, int &f_vertical,
      int &x0, int &y0, int verbose_level);
   int rank(int f_vertical, int x0, int y0, int verbose_level);
   void unrank_coordinates(int rk,
      int &x1, int &y1, int &x2, int &y2,
      int verbose_level);
   int rank_coordinates(int x1, int y1, int x2, int y2,
      int verbose_level);
};




// ##############################################################################
// classification_of_objects_description.cpp
// ##############################################################################




//! description of a classification of objects using class classification_of_obje
cts



class classification_of_objects_description {

public:

   int f_label;
   std::string label;

   int f_save_classification;
   std::string save_prefix;

   int f_max_TDO_depth;
   int max_TDO_depth;
```

```
   int f_classification_prefix;
   std::string classification_prefix;

   int f_save_canonical_labeling;

   int f_save_ago;

   int f_save_transversal;


   classification_of_objects_description();
   ~classification_of_objects_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};

// ##############################################################################
// classification_of_objects_report_options.cpp
// ##############################################################################




//! options for the report for a classification of combinatorial objects



class classification_of_objects_report_options {

public:

   int f_prefix;
   std::string prefix;

   int f_export_flag_orbits;

   int f_show_incidence_matrices;

   int f_show_TDO;

   int f_show_TDA;

   int f_export_group_orbiter;
   int f_export_group_GAP;
```

```
   int f_lex_least;
   std::string lex_least_geometry_builder;

   classification_of_objects_report_options();
   ~classification_of_objects_report_options();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// classification_of_objects.cpp
// ##############################################################################




//! classification of combinatorial objects using a graph-theoretic approach



class classification_of_objects {

public:

   classification_of_objects_description *Descr;

   int f_projective_space;
   //projective_space_with_action *PA;
   geometry::projective_space *P;

   data_structures::data_input_stream *IS;


   data_structures::classify_bitvectors *CB;

   long int *Ago; // [IS->nb_objects_to_test]
   int *F_reject; // [IS->nb_objects_to_test]


   // the classification:

   int nb_orbits; // number of isomorphism types
```

```
int *Idx_transversal; // [nb_orbits]

long int *Ago_transversal; // [nb_orbits]

geometry::object_with_canonical_form **OWCF_transversal; // [nb_orbits]

data_structures::nauty_output **NO_transversal; // [nb_orbits]


data_structures::tally *T_Ago;



classification_of_objects();
~classification_of_objects();
void perform_classification(classification_of_objects_description *Descr,
      int f_projective_space,
      geometry::projective_space *P,
      data_structures::data_input_stream *IS,
      int verbose_level);
void classify_objects_using_nauty(
   int verbose_level);
void save_automorphism_group_order(int verbose_level);
void save_transversal(int verbose_level);
void process_any_object(
      geometry::object_with_canonical_form *OwCF,
      int input_idx, long int &ago, int &f_reject,
      data_structures::nauty_output *&NO,
      int verbose_level);
int process_object(
      geometry::object_with_canonical_form *OwCF,
   long int &ago,
   int &iso_idx_if_found,
   data_structures::nauty_output *&NO,
   int verbose_level);
// returns f_found, which is TRUE if the object is already in the list
void report_summary_of_orbits(
      std::ostream &fp, int verbose_level);
void report_all_isomorphism_types(
      std::ostream &fp, int max_TDO_depth,
      int f_show_incma,
      int verbose_level);
void report_isomorphism_type(
      std::ostream &fp, int i, int max_TDO_depth,
      int f_show_incma,
      int verbose_level);
```

```
   void report_object(std::ostream &fp,
         geometry::object_with_canonical_form *OwCF,
         int object_idx,
         int max_TDO_depth,
         int f_show_incma,
         int verbose_level);


};





// ##############################################################################
// combinatorics_domain.cpp
// ##############################################################################

//! a collection of combinatorial functions

class combinatorics_domain {

public:
   combinatorics_domain();
   ~combinatorics_domain();
   int int_factorial(int a);
   int Kung_mue_i(int *part, int i, int m);
   void partition_dual(int *part, int *dual_part, int n, int verbose_level);
   void make_all_partitions_of_n(int n, int *&Table, int &nb, int verbose_level);
   int count_all_partitions_of_n(int n);
   int partition_first(int *v, int n);
   int partition_next(int *v, int n);
   void partition_print(std::ostream &ost, int *v, int n);
   int int_vec_is_regular_word(int *v, int len, int q);
      // Returns TRUE if the word v of length len is regular, i.~e.
      // lies in an orbit of length $len$ under the action of the cyclic group
      // $C_{len}$ acting on the coordinates.
      // Lueneburg~\cite{Lueneburg87a} p. 118.
      // v is a vector over $\{0, 1, \ldots , q-1\}$
   int int_vec_first_regular_word(int *v, int len, int q);
   int int_vec_next_regular_word(int *v, int len, int q);
   void int_vec_splice(int *v, int *w, int len, int p);
   int is_subset_of(int *A, int sz_A, int *B, int sz_B);
   int set_find(int *elts, int size, int a);
   void set_complement(
```

```
      int *subset, int subset_size, int *complement,
   int &size_complement, int universal_set_size);
void set_complement_lint(
      long int *subset, int subset_size, long int *complement,
   int &size_complement, int universal_set_size);
void set_complement_safe(
      int *subset, int subset_size, int *complement,
   int &size_complement, int universal_set_size);
// subset does not need to be in increasing order
void set_add_elements(int *elts, int &size,
   int *elts_to_add, int nb_elts_to_add);
void set_add_element(int *elts, int &size, int a);
void set_delete_elements(int *elts, int &size,
   int *elts_to_delete, int nb_elts_to_delete);
void set_delete_element(int *elts, int &size, int a);
int compare_lexicographically(
      int a_len, long int *a, int b_len, long int *b);
long int int_n_choose_k(int n, int k);
void make_t_k_incidence_matrix(
      int v, int t, int k, int &m, int &n, int *&M,
   int verbose_level);
void print_k_subsets_by_rank(std::ostream &ost, int v, int k);
int f_is_subset_of(
      int v, int t, int k,
      int rk_t_subset, int rk_k_subset);
int rank_subset(int *set, int sz, int n);
void rank_subset_recursion(int *set, int sz, int n, int a0, int &r);
void unrank_subset(int *set, int &sz, int n, int r);
void unrank_subset_recursion(int *set, int &sz, int n, int a0, int &r);
int rank_k_subset(int *set, int n, int k);
void unrank_k_subset(int rk, int *set, int n, int k);
void unrank_k_subset_and_complement(int rk, int *set, int n, int k);
int first_k_subset(int *set, int n, int k);
int next_k_subset(int *set, int n, int k);
int next_k_subset_at_level(int *set, int n, int k, int backtrack_level);
void subset_permute_up_front(int n, int k, int *set, int *k_subset_idx,
   int *permuted_set);
int ordered_pair_rank(int i, int j, int n);
void ordered_pair_unrank(int rk, int &i, int &j, int n);
void set_partition_4_into_2_unrank(int rk, int *v);
int set_partition_4_into_2_rank(int *v);
int unordered_triple_pair_rank(int i, int j, int k, int l, int m, int n);
void unordered_triple_pair_unrank(int rk, int &i, int &j, int &k,
   int &l, int &m, int &n);
long int ij2k_lint(long int i, long int j, long int n);
void k2ij_lint(long int k, long int & i, long int & j, long int n);
int ij2k(int i, int j, int n);
```

```
void k2ij(int k, int & i, int & j, int n);
int ijk2h(int i, int j, int k, int n);
void h2ijk(int h, int &i, int &j, int &k, int n);
void random_permutation(int *random_permutation, long int n);
void perm_move(int *from, int *to, long int n);
void perm_identity(int *a, long int n);
int perm_is_identity(int *a, long int n);
void perm_elementary_transposition(int *a, long int n, int f);
void perm_mult(int *a, int *b, int *c, long int n);
void perm_conjugate(int *a, int *b, int *c, long int n);
// c := a^b = b^-1 * a * b
void perm_inverse(int *a, int *b, long int n);
// b := a^-1
void perm_raise(int *a, int *b, int e, long int n);
// b := a^e (e >= 0)
void perm_direct_product(long int n1, long int n2,
      int *perm1, int *perm2, int *perm3);
void perm_print_list(std::ostream &ost, int *a, int n);
void perm_print_list_offset(std::ostream &ost, int *a, int n, int offset);
void perm_print_product_action(std::ostream &ost, int *a, int m_plus_n, int m,
   int offset, int f_cycle_length);
void perm_print(std::ostream &ost, int *a, int n);
void perm_print_with_print_point_function(
      std::ostream &ost,
      int *a, int n,
      void (*point_label)(
            std::stringstream &sstr, long int pt, void *data),
      void *point_label_data);
void perm_print_with_cycle_length(std::ostream &ost, int *a, int n);
void perm_print_counting_from_one(std::ostream &ost, int *a, int n);
void perm_print_offset(std::ostream &ost,
   int *a, int n,
   int offset,
   int f_print_cycles_of_length_one,
   int f_cycle_length,
   int f_max_cycle_length,
   int max_cycle_length,
   int f_orbit_structure,
   void (*point_label)(std::stringstream &sstr, long int pt, void *data),
   void *point_label_data);
void perm_cycle_type(int *perm, long int degree, int *cycles, int &nb_cycles);
int perm_order(int *a, long int n);
int perm_signum(int *perm, long int n);
int is_permutation(int *perm, long int n);
int is_permutation_lint(long int *perm, long int n);
void first_lehmercode(int n, int *v);
int next_lehmercode(int n, int *v);
```

```
void lehmercode_to_permutation(int n, int *code, int *perm);
int disjoint_binary_representation(int u, int v);
int hall_test(int *A, int n, int kmax, int *memo, int verbose_level);
int philip_hall_test(int *A, int n, int k, int *memo, int verbose_level);
int philip_hall_test_dual(
      int *A, int n, int k, int *memo, int verbose_level);
void print_01_matrix_with_stars(
      std::ostream &ost, int *A, int m, int n);
void print_int_matrix(std::ostream &ost, int *A, int m, int n);
int create_roots_H4(field_theory::finite_field *F, int *roots);
long int generalized_binomial(int n, int k, int q);
void print_tableau(int *Tableau, int l1, int l2,
   int *row_parts, int *col_parts);
int ijk_rank(int i, int j, int k, int n);
void ijk_unrank(int &i, int &j, int &k, int n, int rk);
long int largest_binomial2_below(int a2);
long int largest_binomial3_below(int a3);
long int binomial2(int a);
long int binomial3(int a);
int minus_one_if_positive(int i);
void make_partitions(int n, int *Part, int cnt);
int count_partitions(int n);
int next_partition(int n, int *part);
long int binomial_lint(int n, int k);
void binomial(
      ring_theory::longinteger_object &a,
      int n, int k, int verbose_level);
void binomial_with_table(
      ring_theory::longinteger_object &a,
      int n, int k);
void size_of_conjugacy_class_in_sym_n(
      ring_theory::longinteger_object &a, int n, int *part);
void q_binomial_with_table(ring_theory::longinteger_object &a,
   int n, int k, int q, int verbose_level);
void q_binomial(
      ring_theory::longinteger_object &a,
   int n, int k, int q, int verbose_level);
void q_binomial_no_table(
      ring_theory::longinteger_object &a,
   int n, int k, int q, int verbose_level);
void krawtchouk_with_table(ring_theory::longinteger_object &a,
   int n, int q, int k, int x);
void krawtchouk(
      ring_theory::longinteger_object &a,
      int n, int q, int k, int x);
void do_tdo_refinement(
      tdo_refinement_description *Descr, int verbose_level);
```

```
void do_tdo_print(
      std::string &fname, int verbose_level);
void make_elementary_symmetric_functions(
      int n, int k_max, int verbose_level);
void Dedekind_numbers(
      int n_min, int n_max, int q_min, int q_max,
      int verbose_level);
void convert_stack_to_tdo(
      std::string &stack_fname, int verbose_level);
void do_parameters_maximal_arc(
      int q, int r, int verbose_level);
void do_parameters_arc(
      int q, int s, int r, int verbose_level);
void do_read_poset_file(
      std::string &fname,
      int f_grouping, double x_stretch, int verbose_level);
// creates a layered graph file from a text file
// which was created by DISCRETA/sgls2.cpp
void do_make_tree_of_all_k_subsets(
      int n, int k, int verbose_level);
void create_random_permutation(int deg,
      std::string &fname_csv, int verbose_level);
void create_random_k_subsets(int n, int k, int nb,
      std::string &fname_csv, int verbose_level);
void compute_incidence_matrix(
      int v, int b, int k, long int *Blocks_coded,
      int *&M, int verbose_level);
void compute_incidence_matrix_from_sets(
      int v, int b, long int *Sets_coded,
      int *&M,
      int verbose_level);
void compute_blocks_from_coding(
      int v, int b, int k, long int *Blocks_coded,
      int *&Blocks, int verbose_level);
void compute_blocks_from_incma(
      int v, int b, int k, int *incma,
      int *&Blocks, int verbose_level);
void refine_the_partition(
      int v, int k, int b, long int *Blocks_coded,
      int &b_reduced,
      int verbose_level);
void create_incidence_matrix_of_graph(int *Adj, int n,
      int *&M, int &nb_rows, int &nb_cols,
      int verbose_level);
void free_global_data();
void free_tab_q_binomials();
void create_wreath_product_design(int n, int k,
```

```
        long int *&Blocks, long int &nb_blocks,
        int verbose_level);

};


// ##############################################################################
// domino_assignment.cpp:
// ##############################################################################

// the dimensions are (<D>+1) * <s>   x <D>*<s>
// so, for D=7 we would get
// s=4:   32x28
// s=5:   40x35

//! compute a domino portrait using an optimization algorithm

class domino_assignment {
public:
   int D;
   int s;
   int size_dom;
   int tot_dom;

   int M; // number of rows  = (D + 1) * s
   int N; // number of columns = D * s

   int *ij_posi; // [M * N * 2];
      // ij_posi[(i * N + j) * 2 + 0] = i
      // ij_posi[(i * N + j) * 2 + 1] = j

   int *assi; // [tot_dom * 5];
      // 0: m
      // 1: n
      // 2: o = orientation
      // 3: i
      // 4: j
      // where (i,j) is the place of the top or left half of the domino

   int *broken_dom; // [M * N]
      // broken_dom[i * n + j] is the index in the assi array
      // of the domino piece covering the place (i,j)
   int *matching; // [M * N]
      // matching[i * N + j] tells the direction
      // of the second half of the domino
   int *A; // [M * N], the domino matrix
   int *mphoto; // [M * N], the photo matrix
```

```
int *North; // [M * N]
int *South; // [M * N]
int *West; // [M * N]
int *East; // [M * N]

int brake_cnt;
int *brake; // [tot_dom], used as [brake_cnt]

int nb_changes;

std::vector<domino_change> Changes;


domino_assignment();
~domino_assignment();
void stage0(int verbose_level);
void stage1(int verbose_level);
void stage2(int verbose_level);
void initialize_assignment(int D, int s, int verbose_level);
void init_matching(int verbose_level);
int cost_function();
int compute_cost_of_one_piece(int idx);
int compute_cost_of_one_piece_directly(
      int m, int n, int o, int i, int j);
int my_distance(int a, int b);
void compute_domino_matrix(int depth);
void move(domino_assignment *To);
void draw_domino_matrix(std::string &fname,
      int depth,
      int f_has_cost, int cost,
      graphics::layered_graph_draw_options *Draw_options,
      int verbose_level);
void draw_domino_matrix2(std::string &fname,
      int f_has_cost, int cost,
   int f_frame, int f_grid, int f_B, int *B,
   int f_numbers, int f_gray,
   graphics::layered_graph_draw_options *Draw_options,
   int verbose_level);
void read_photo(std::string &photo_fname, int verbose_level);
void scale_photo(double *dphoto, int verbose_level);
void do_flip_recorded(int f2, int verbose_level);
void do_flip(int f2, int verbose_level);
void flip_each(int verbose_level);
void flip_randomized(int verbose_level);
void do_swap_recorded(int s1, int s2, int verbose_level);
void do_swap(int s1, int s2, int verbose_level);
```

```
   int do_flipswap(int f2);
   void swap_randomized(int verbose_level);
   void swap_each(int verbose_level);
   void do_horizontal_rotate(int ro, int verbose_level);
   void do_vertical_rotate(int ro, int verbose_level);
   int modify_matching(
         int idx_first_broken,
         int ass_m, int ass_n,
         int ass_o, int ass_i, int ass_j,
         int verbose_level);
   void follow_the_matching(
         int l, int *used, int *reached,
         int *list, int *length, int *prec,
         int verbose_level);
   int find_match(int l,
      int *reached1, int *list1, int *length1, int *prec1,
      int *reached2, int *list2, int *length2, int *prec2,
      int verbose_level);
   int breadth_search(int l, int *used, int *reached,
         int *list, int *length, int *prec,
         int verbose_level);
   void rotate_once(int ro, int verbose_level);
   void rotate_randomized(int verbose_level);
   void do_horizontal_shift(int ro, int verbose_level);
   void do_vertical_shift(int ro, int verbose_level);
   void shift_once(int ro, int verbose_level);
   void shift_once_randomized(int verbose_level);
   void shift_randomized(int verbose_level);
   void flip_after_shift(int verbose_level);
   void print_matching(std::ostream &ost);
   void print(std::ostream &ost);
   void prepare_latex(std::string &photo_label, int verbose_level);
   void record_flip(int idx, int verbose_level);
   void record_swap(int s1, int s2, int verbose_level);
   void record_matching(int verbose_level);
   void drop_changes_to(int nb_changes_to_drop_to, int verbose_level);
   void classify_changes_by_type(int verbose_level);
   void get_cost_function(int *&Cost, int &len, int verbose_level);
};

// ##############################################################################
// domino_change.cpp:
// ##############################################################################


//! utility class for the domino portrait algorithm
```

```
class domino_change {
public:
   int type_of_change;
   int cost_after_change;

   domino_change();
   ~domino_change();
   void init(domino_assignment *DA,
         int type_of_change, int verbose_level);
};




// ##############################################################################
// encoded_combinatorial_object.cpp
// ##############################################################################

//! encoding of combinatorial object for use with nauty


class encoded_combinatorial_object {

private:
   int *Incma;

public:
   int nb_rows0;
   int nb_cols0;

   int nb_flags;
   int nb_rows;
   int nb_cols;
   int *partition;
   int canonical_labeling_len; // = nb_rows + nb_cols

   encoded_combinatorial_object();
   ~encoded_combinatorial_object();
   void init_everything(int nb_rows, int nb_cols,
         int *Incma, int *partition,
         int verbose_level);
   void init(int nb_rows, int nb_cols, int verbose_level);
   int *get_Incma();
   void set_incidence_ij(int i, int j);
   int get_incidence_ij(int i, int j);
   void set_incidence(int a);
   void init_canonical_form(
```

```
        encoded_combinatorial_object *Enc,
        data_structures::nauty_output *NO,
        int verbose_level);
    void print_incma();
    void print_partition();
    void compute_canonical_incma(int *canonical_labeling,
        int *&Incma_out, int verbose_level);
    void compute_canonical_form(
        data_structures::bitvector *&Canonical_form,
        int *canonical_labeling, int verbose_level);
    void incidence_matrix_projective_space_top_left(
        geometry::projective_space *P, int verbose_level);
    void extended_incidence_matrix_projective_space_top_left(
        geometry::projective_space *P, int verbose_level);
    void canonical_form_given_canonical_labeling(
        int *canonical_labeling,
        data_structures::bitvector *&B,
        int verbose_level);
    void latex_set_system_by_columns(std::ostream &ost,
        int verbose_level);
    void latex_set_system_by_rows(std::ostream &ost,
        int verbose_level);
    void latex_incma(std::ostream &ost, int verbose_level);
    void latex_TDA(std::ostream &ost,
        int nb_orbits, int *orbit_first, int *orbit_len, int *orbit,
        int verbose_level);
    void latex_TDA_with_labels(std::ostream &ost,
        int nb_orbits, int *orbit_first, int *orbit_len, int *orbit,
        int verbose_level);
    void latex_canonical_form(std::ostream &ost,
        data_structures::nauty_output *NO,
        int verbose_level);
    void apply_canonical_labeling(int *&Inc2,
        data_structures::nauty_output *NO);
    void apply_canonical_labeling_and_get_flags(int *&Inc2,
        int *&Flags, int &nb_flags_counted,
        data_structures::nauty_output *NO);
    void latex_canonical_form_with_labels(std::ostream &ost,
        data_structures::nauty_output *NO,
        std::string *row_labels,
        std::string *col_labels,
        int verbose_level);

};
```

```
// ##############################################################################
// geo_parameter.cpp
// ##############################################################################


#define MODE_UNDEFINED 0
#define MODE_SINGLE 1
#define MODE_STACK 2

#define UNKNOWNTYPE 0
#define POINTTACTICAL 1
#define BLOCKTACTICAL 2
#define POINTANDBLOCKTACTICAL 3

#define FUSE_TYPE_NONE 0
#define FUSE_TYPE_SIMPLE 1
#define FUSE_TYPE_DOUBLE 2
//#define FUSE_TYPE_MULTI 3
//#define FUSE_TYPE_TDO 4

//! decomposition stack of a linear space or incidence geometry



class geo_parameter {
public:
   int decomposition_type;
   int fuse_type;
   int v, b;

   int mode;
   std::string label;

   // for MODE_SINGLE
   int nb_V, nb_B;
   int *V, *B;
   int *scheme;
   int *fuse;

   // for MODE_STACK
   int nb_parts, nb_entries;

   int *part;
   int *entries;
```

```
int part_nb_alloc;
int entries_nb_alloc;


int lambda_level;
int row_level, col_level;
int extra_row_level, extra_col_level;

geo_parameter();
~geo_parameter();
void append_to_part(int a);
void append_to_entries(int a1, int a2, int a3, int a4);
void write(std::ofstream &aStream, std::string &label);
void write_mode_single(
      std::ofstream &aStream,
      std::string &label);
void write_mode_stack(
      std::ofstream &aStream,
      std::string &label);
void convert_single_to_stack(int verbose_level);
int partition_number_row(int row_idx);
int partition_number_col(int col_idx);
int input(std::ifstream &aStream);
int input_mode_single(std::ifstream &aStream);
int input_mode_stack(
      std::ifstream &aStream, int verbose_level);
void init_tdo_scheme(
      tdo_scheme_synthetic &G, int verbose_level);
void print_schemes(tdo_scheme_synthetic &G);
void print_schemes_tex(tdo_scheme_synthetic &G);
void print_scheme_tex(
      std::ostream &ost,
      tdo_scheme_synthetic &G, int h);
void print_C_source();
void convert_single_to_stack_fuse_simple_pt(int verbose_level);
void convert_single_to_stack_fuse_simple_bt(int verbose_level);
void convert_single_to_stack_fuse_double_pt(int verbose_level);
void cut_off_two_lines(geo_parameter &GP2,
   int *&part_relabel, int *&part_length,
   int verbose_level);
void cut_off(geo_parameter &GP2, int w,
   int *&part_relabel, int *&part_length,
   int verbose_level);
void copy(geo_parameter &GP2);
void print_schemes();
int tdo_scheme_get_row_class_length_fused(
      tdo_scheme_synthetic &G,
```

```
        int h, int class_first, int class_len);
   int tdo_scheme_get_col_class_length_fused(
        tdo_scheme_synthetic &G,
        int h, int class_first, int class_len);
};



// ##############################################################################
// pentomino_puzzle.cpp
// ##############################################################################


#define NB_PIECES 18



//! generate all solutions of the pentomino puzzle


class pentomino_puzzle {

   public:
   int *S[NB_PIECES];
   int S_length[NB_PIECES];
   int *O[NB_PIECES];
   int O_length[NB_PIECES];
   int *T[NB_PIECES];
   int T_length[NB_PIECES];
   int *R[NB_PIECES];
   int R_length[NB_PIECES];
   int Rotate[4 * 25];
   int Rotate6[4 * 36];
   int var_start[NB_PIECES + 1];
   int var_length[NB_PIECES + 1];


   void main(int verbose_level);
   int has_interlocking_Ps(long int *set);
   int has_interlocking_Pprime(long int *set);
   int has_interlocking_Ls(long int *set);
   int test_if_interlocking_Ps(int a1, int a2);
   int has_interlocking_Lprime(long int *set);
   int test_if_interlocking_Ls(int a1, int a2);
   int number_of_pieces_of_type(int t, long int *set);
   int does_it_contain_an_I(long int *set);
   void decode_assembly(long int *set);
```

```
   // input set[5]
   void decode_piece(int j, int &h, int &r, int &t);
   // h is the kind of piece
   // r is the rotation index
   // t is the translation index
   // to get the actual rotation and translation, use
   // R[h][r] and T[h][t].
   int code_piece(int h, int r, int t);
   void draw_it(std::ostream &ost, long int *sol);
   void compute_image_function(
         data_structures::set_of_sets *S,
         int elt_idx,
         int gen_idx, int &idx_of_image, int verbose_level);
   void turn_piece(int &h, int &r, int &t, int verbose_level);
   void flip_piece(int &h, int &r, int &t, int verbose_level);
   void setup_pieces();
   void setup_rotate();
   void setup_var_start();
   void make_coefficient_matrix(solvers::diophant *D);

};



// ##############################################################################
// polynomial_function_domain.cpp
// ##############################################################################

//! polynomial expressions for functions from a finite field to itself

class polynomial_function_domain {

public:
   field_theory::finite_field *Fq; // the field Fq
   int q;

   int n;
   int max_degree; // n * (q - 1)

   int Q; // q^n = number of inputs to the function.

   ring_theory::homogeneous_polynomial_domain *Poly;
      // Poly[i] = polynomial of degree i in n + 1 variables.
      // i = 1,..,max_degree
   int **A_poly; // [1..max_degree][Poly[i].get_nb_monomials()]
   int **B_poly; // [1..max_degree][Poly[i].get_nb_monomials()]
   int **C_poly; // [1..max_degree][Poly[i].get_nb_monomials()]
```

```
   int *Kernel;
   int dim_kernel;


   long int *affine_points; // [Q]
      // affine_points[i] = PG_rank of affine point[i]



   int *v; // [n]
   int *v1; // [n + 1]
   int *w; // [n]
   int *f; // [Q]
   int *f2; // [Q]


   polynomial_function_domain();
   ~polynomial_function_domain();
   void init(
         field_theory::finite_field *Fq,
         int n, int verbose_level);
   void setup_polynomial_rings(int verbose_level);
   void compute_polynomial_representation(
         int *func, int *coeff, int verbose_level);
   void evaluate_projectively(int *coeff, int *f);
   void evaluate(int *coeff, int *f);
   void raise(int *in, int *out);
   void multiply_i_times_j(
         int i, int j,
         int *A_eqn, int *B_eqn, int *C_eqn,
      int verbose_level);
   void algebraic_normal_form(int *func, int len,
         int *&coeff, int &nb_coeff,
         int verbose_level);
};




// ##############################################################################
// tdo_data.cpp TDO parameter refinement
// ##############################################################################

//! a utility class related to the class tdo_scheme

class tdo_data {
```

```
public:
   int *types_first;
   int *types_len;
   int *only_one_type;
   int nb_only_one_type;
   int *multiple_types;
   int nb_multiple_types;
   int *types_first2;
   solvers::diophant *D1;
   solvers::diophant *D2;

   tdo_data();
   ~tdo_data();
   void free();
   void allocate(int R);
   int solve_first_system(int verbose_level,
      int *&line_types, int &nb_line_types,
      int &line_types_allocated);
   void solve_second_system_omit(int verbose_level,
      int *classes_len,
      int *&line_types, int &nb_line_types,
      int *&distributions, int &nb_distributions,
      int omit);
   void solve_second_system_with_help(int verbose_level,
      int f_use_mckay_solver, int f_once,
      int *classes_len, int f_scale, int scaling,
      int *&line_types, int &nb_line_types,
      int *&distributions, int &nb_distributions,
      int cnt_second_system, solution_file_data *Sol);
   void solve_second_system_from_file(int verbose_level,
      int *classes_len, int f_scale, int scaling,
      int *&line_types, int &nb_line_types,
      int *&distributions, int &nb_distributions,
      std::string &solution_file_name);
   void solve_second_system(int verbose_level,
      int f_use_mckay_solver, int f_once,
      int *classes_len, int f_scale, int scaling,
      int *&line_types, int &nb_line_types,
      int *&distributions, int &nb_distributions);
};

// ##############################################################################
// tdo_refinement_description.cpp
// ##############################################################################
```

```
//! input data for the parameter refinement of a linear space

class tdo_refinement_description {
   public:

   int f_lambda3;
   int lambda3, block_size;
   int f_scale;
   int scaling;
   int f_range;
   int range_first, range_len;
   int f_select;
   std::string select_label;
   int f_omit1;
   int omit1;
   int f_omit2;
   int omit2;
   int f_D1_upper_bound_x0;
   int D1_upper_bound_x0;
   int f_reverse;
   int f_reverse_inverse;
   int f_use_packing_numbers;
   int f_dual_is_linear_space;
   int f_do_the_geometric_test;
   int f_once;
   int f_use_mckay_solver;
   int f_input_file;
   std::string fname_in;

   solution_file_data *Sol;

   tdo_refinement_description();
   ~tdo_refinement_description();
   int read_arguments(int argc, std::string *argv, int verbose_level);
   void print();

};


// ##############################################################################
// tdo_refinement.cpp
// ##############################################################################



//! refinement of the parameters of a linear space
```

```
class tdo_refinement {
   public:

   int t0;
   int cnt;

   tdo_refinement_description *Descr;

   std::string fname;
   std::string fname_out;



   geo_parameter GP;

   geo_parameter GP2;



   int f_doit;
   int nb_written, nb_written_tactical, nb_tactical;
   int cnt_second_system;

   tdo_refinement();
   ~tdo_refinement();
   void init(
         tdo_refinement_description *Descr,
         int verbose_level);
   void main_loop(int verbose_level);
   void do_it(
         std::ofstream &g, int verbose_level);
   void do_row_refinement(
         std::ofstream &g,
         tdo_scheme_synthetic &G,
         data_structures::partitionstack &P,
         int verbose_level);
   void do_col_refinement(
         std::ofstream &g,
         tdo_scheme_synthetic &G,
         data_structures::partitionstack &P,
         int verbose_level);
   void do_all_row_refinements(
         std::string &label_in, std::ofstream &g,
         tdo_scheme_synthetic &G,
      int *point_types, int nb_point_types, int point_type_len,
      int *distributions, int nb_distributions, int &nb_tactical,
      int verbose_level);
```

```
   void do_all_column_refinements(
         std::string &label_in, std::ofstream &g,
         tdo_scheme_synthetic &G,
      int *line_types, int nb_line_types, int line_type_len,
      int *distributions, int nb_distributions, int &nb_tactical,
      int verbose_level);
   int do_row_refinement(
         int t,
         std::string &label_in,
         std::ofstream &g,
         tdo_scheme_synthetic &G,
      int *point_types, int nb_point_types, int point_type_len,
      int *distributions, int nb_distributions,
      int verbose_level);
      // returns TRUE or FALSE depending on whether the
      // refinement gave a tactical decomposition
   int do_column_refinement(
         int t, std::string &label_in,
         std::ofstream &g,
         tdo_scheme_synthetic &G,
      int *line_types, int nb_line_types, int line_type_len,
      int *distributions, int nb_distributions,
      int verbose_level);
      // returns TRUE or FALSE depending on whether the
      // refinement gave a tactical decomposition
};



// ##############################################################################
// tdo_scheme_compute.cpp
// ##############################################################################

//! tactical decomposition of an incidence structure obtained by refinement

class tdo_scheme_compute {

public:

   encoded_combinatorial_object *Enc;
   geometry::decomposition *Decomp;



   tdo_scheme_compute();
   ~tdo_scheme_compute();
   void init(encoded_combinatorial_object *Enc,
```

```
        int max_depth,
        int verbose_level);
   void print_schemes(std::ostream &ost);

};


// ##############################################################################
// tdo_scheme_synthetic.cpp
// ##############################################################################



#define NUMBER_OF_SCHEMES 5
#define ROW_SCHEME 0
#define COL_SCHEME 1
#define LAMBDA_SCHEME 2
#define EXTRA_ROW_SCHEME 3
#define EXTRA_COL_SCHEME 4


//! internal class related to class tdo_data


struct solution_file_data {
   int nb_solution_files;
   std::vector<int> system_no;
   std::vector<std::string> solution_file;
};

//! canonical tactical decomposition of an incidence structure

class tdo_scheme_synthetic {

public:

   // the following is needed by the TDO process:
   // allocated in init_partition_stack
   // freed in exit_partition_stack

   //partition_backtrack PB;

   data_structures::partitionstack *P;

   int part_length;
   int *part;
   int nb_entries;
```

```
int *entries;
int row_level;
int col_level;
int lambda_level;
int extra_row_level;
int extra_col_level;

int mn; // m + n
int m; // # of rows
int n; // # of columns

int level[NUMBER_OF_SCHEMES];
int *row_classes[NUMBER_OF_SCHEMES], nb_row_classes[NUMBER_OF_SCHEMES];
int *col_classes[NUMBER_OF_SCHEMES], nb_col_classes[NUMBER_OF_SCHEMES];
int *row_class_index[NUMBER_OF_SCHEMES];
int *col_class_index[NUMBER_OF_SCHEMES];
int *row_classes_first[NUMBER_OF_SCHEMES];
int *row_classes_len[NUMBER_OF_SCHEMES];
int *row_class_no[NUMBER_OF_SCHEMES];
int *col_classes_first[NUMBER_OF_SCHEMES];
int *col_classes_len[NUMBER_OF_SCHEMES];
int *col_class_no[NUMBER_OF_SCHEMES];

int *the_row_scheme;
int *the_col_scheme;
int *the_extra_row_scheme;
int *the_extra_col_scheme;
int *the_row_scheme_cur; // [m * nb_col_classes[ROW_SCHEME]]
int *the_col_scheme_cur; // [n * nb_row_classes[COL_SCHEME]]
int *the_extra_row_scheme_cur; // [m * nb_col_classes[EXTRA_ROW_SCHEME]]
int *the_extra_col_scheme_cur; // [n * nb_row_classes[EXTRA_COL_SCHEME]]

// end of TDO process data

tdo_scheme_synthetic();
~tdo_scheme_synthetic();

void init_part_and_entries(
      int *part, int *entries, int verbose_level);
void init_part_and_entries_int(
      int *part, int *entries, int verbose_level);
void init_TDO(int *Part, int *Entries,
   int Row_level, int Col_level,
   int Extra_row_level, int Extra_col_level,
   int Lambda_level, int verbose_level);
void exit_TDO();
void init_partition_stack(int verbose_level);
```

```
void exit_partition_stack();
void get_partition(int h, int l, int verbose_level);
void free_partition(int h);
void complete_partition_info(int h, int verbose_level);
void get_row_or_col_scheme(int h, int l, int verbose_level);
void get_column_split_partition(int verbose_level,
      data_structures::partitionstack &P);
void get_row_split_partition(int verbose_level,
      data_structures::partitionstack &P);
void print_all_schemes();
void print_scheme(int h, int verbose_level);
void print_scheme_tex(std::ostream &ost, int h);
void print_scheme_tex_fancy(
      std::ostream &ost,
      int h, int f_label, std::string &label);
void compute_whether_first_inc_must_be_moved(
      int *f_first_inc_must_be_moved, int verbose_level);
int count_nb_inc_from_row_scheme(int verbose_level);
int count_nb_inc_from_extra_row_scheme(int verbose_level);


int geometric_test_for_row_scheme(
      data_structures::partitionstack &P,
   int *point_types, int nb_point_types, int point_type_len,
   int *distributions, int nb_distributions,
   int f_omit1, int omit1, int verbose_level);
int geometric_test_for_row_scheme_level_s(
      data_structures::partitionstack &P, int s,
   int *point_types, int nb_point_types, int point_type_len,
   int *distribution,
   int *non_zero_blocks, int nb_non_zero_blocks,
   int f_omit1, int omit1,
   int verbose_level);


int refine_rows(int verbose_level,
   int f_use_mckay, int f_once,
   data_structures::partitionstack &P,
   int *&point_types, int &nb_point_types, int &point_type_len,
   int *&distributions, int &nb_distributions,
   int &cnt_second_system, solution_file_data *Sol,
   int f_omit1, int omit1, int f_omit2, int omit2,
   int f_use_packing_numbers,
   int f_dual_is_linear_space,
   int f_do_the_geometric_test);
int refine_rows_easy(int verbose_level,
   int *&point_types, int &nb_point_types, int &point_type_len,
```

```
   int *&distributions, int &nb_distributions,
   int &cnt_second_system);
int refine_rows_hard(
      data_structures::partitionstack &P,
      int verbose_level,
   int f_use_mckay, int f_once,
   int *&point_types, int &nb_point_types, int &point_type_len,
   int *&distributions, int &nb_distributions,
   int &cnt_second_system,
   int f_omit1, int omit1, int f_omit, int omit,
   int f_use_packing_numbers, int f_dual_is_linear_space);
void row_refinement_L1_L2(
      data_structures::partitionstack &P,
      int f_omit, int omit,
   int &L1, int &L2, int verbose_level);
int tdo_rows_setup_first_system(
      int verbose_level,
   tdo_data &T, int r,
   data_structures::partitionstack &P,
   int f_omit, int omit,
   int *&point_types, int &nb_point_types);
int tdo_rows_setup_second_system(
      int verbose_level,
   tdo_data &T,
   data_structures::partitionstack &P,
   int f_omit, int omit,
   int f_use_packing_numbers,
   int f_dual_is_linear_space,
   int *&point_types, int &nb_point_types);
int tdo_rows_setup_second_system_eqns_joining(
      int verbose_level,
   tdo_data &T,
   data_structures::partitionstack &P,
   int f_omit, int omit, int f_dual_is_linear_space,
   int *point_types, int nb_point_types,
   int eqn_offset);
int tdo_rows_setup_second_system_eqns_counting(
      int verbose_level,
   tdo_data &T,
   data_structures::partitionstack &P,
   int f_omit, int omit,
   int *point_types, int nb_point_types,
   int eqn_offset);
int tdo_rows_setup_second_system_eqns_packing(
      int verbose_level,
   tdo_data &T,
   data_structures::partitionstack &P,
```

```
   int f_omit, int omit,
   int *point_types, int nb_point_types,
   int eqn_start, int &nb_eqns_used);

int refine_columns(
      int verbose_level, int f_once,
      data_structures::partitionstack &P,
   int *&line_types, int &nb_line_types, int &line_type_len,
   int *&distributions, int &nb_distributions,
   int &cnt_second_system, solution_file_data *Sol,
   int f_omit1, int omit1, int f_omit, int omit,
   int f_D1_upper_bound_x0, int D1_upper_bound_x0,
   int f_use_mckay_solver,
   int f_use_packing_numbers);
int refine_cols_hard(
      data_structures::partitionstack &P,
   int verbose_level, int f_once,
   int *&line_types, int &nb_line_types, int &line_type_len,
   int *&distributions, int &nb_distributions,
   int &cnt_second_system, solution_file_data *Sol,
   int f_omit1, int omit1, int f_omit, int omit,
   int f_D1_upper_bound_x0, int D1_upper_bound_x0,
   int f_use_mckay_solver,
   int f_use_packing_numbers);
void column_refinement_L1_L2(
      data_structures::partitionstack &P,
   int f_omit, int omit,
   int &L1, int &L2, int verbose_level);
int tdo_columns_setup_first_system(
      int verbose_level,
   tdo_data &T, int r,
   data_structures::partitionstack &P,
   int f_omit, int omit,
   int *&line_types, int &nb_line_types);
int tdo_columns_setup_second_system(
      int verbose_level,
   tdo_data &T,
   data_structures::partitionstack &P,
   int f_omit, int omit,
   int f_use_packing_numbers,
   int *&line_types, int &nb_line_types);
int tdo_columns_setup_second_system_eqns_joining(
      int verbose_level,
   tdo_data &T,
   data_structures::partitionstack &P,
   int f_omit, int omit,
   int *line_types, int nb_line_types,
```

```
   int eqn_start);
void tdo_columns_setup_second_system_eqns_counting(
      int verbose_level,
   tdo_data &T,
   data_structures::partitionstack &P,
   int f_omit, int omit,
   int *line_types, int nb_line_types,
   int eqn_start);
int tdo_columns_setup_second_system_eqns_upper_bound(
      int verbose_level,
   tdo_data &T,
   data_structures::partitionstack &P,
   int f_omit, int omit,
   int *line_types, int nb_line_types,
   int eqn_start, int &nb_eqns_used);


int td3_refine_rows(
      int verbose_level, int f_once,
   int lambda3, int block_size,
   int *&point_types, int &nb_point_types, int &point_type_len,
   int *&distributions, int &nb_distributions);
int td3_rows_setup_first_system(
      int verbose_level,
   int lambda3, int block_size, int lambda2,
   tdo_data &T, int r,
   data_structures::partitionstack &P,
   int &nb_vars,int &nb_eqns,
   int *&point_types, int &nb_point_types);
int td3_rows_setup_second_system(
      int verbose_level,
   int lambda3, int block_size, int lambda2,
   tdo_data &T,
   int nb_vars, int &Nb_vars, int &Nb_eqns,
   int *&point_types, int &nb_point_types);
int td3_rows_counting_flags(
      int verbose_level,
   int lambda3, int block_size, int lambda2, int &S,
   tdo_data &T,
   int nb_vars, int Nb_vars,
   int *&point_types, int &nb_point_types, int eqn_offset);
int td3_refine_columns(
      int verbose_level, int f_once,
   int lambda3, int block_size,
   int f_scale, int scaling,
   int *&line_types, int &nb_line_types, int &line_type_len,
   int *&distributions, int &nb_distributions);
```

```
int td3_columns_setup_first_system(
      int verbose_level,
   int lambda3, int block_size, int lambda2,
   tdo_data &T, int r,
   data_structures::partitionstack &P,
   int &nb_vars, int &nb_eqns,
   int *&line_types, int &nb_line_types);
int td3_columns_setup_second_system(
      int verbose_level,
   int lambda3, int block_size, int lambda2, int f_scale, int scaling,
   tdo_data &T,
   int nb_vars, int &Nb_vars, int &Nb_eqns,
   int *&line_types, int &nb_line_types);
int td3_columns_triples_same_class(
      int verbose_level,
   int lambda3, int block_size,
   tdo_data &T,
   int nb_vars, int Nb_vars,
   int *&line_types, int &nb_line_types, int eqn_offset);
int td3_columns_pairs_same_class(
      int verbose_level,
   int lambda3, int block_size, int lambda2,
   tdo_data &T,
   int nb_vars, int Nb_vars,
   int *&line_types, int &nb_line_types, int eqn_offset);
int td3_columns_counting_flags(
      int verbose_level,
   int lambda3, int block_size, int lambda2, int &S,
   tdo_data &T,
   int nb_vars, int Nb_vars,
   int *&line_types, int &nb_line_types, int eqn_offset);
int td3_columns_lambda2_joining_pairs_from_different_classes(
   int verbose_level,
   int lambda3, int block_size, int lambda2,
   tdo_data &T,
   int nb_vars, int Nb_vars,
   int *&line_types, int &nb_line_types, int eqn_offset);
int td3_columns_lambda3_joining_triples_2_1(
      int verbose_level,
   int lambda3, int block_size, int lambda2,
   tdo_data &T,
   int nb_vars, int Nb_vars,
   int *&line_types, int &nb_line_types, int eqn_offset);
int td3_columns_lambda3_joining_triples_1_1_1(
      int verbose_level,
   int lambda3, int block_size, int lambda2,
   tdo_data &T,
```

```
      int nb_vars, int Nb_vars,
      int *&line_types, int &nb_line_types, int eqn_offset);


};





}}}


#endif /* ORBITER_SRC_LIB_FOUNDATIONS_COMBINATORICS_COMBINATORICS_H_ */
```

3.6 Cryptography

```
/*
 * cryptography.h
 *
 *  Created on: Nov 4, 2020
 *      Author: betten
 */

#ifndef SRC_LIB_FOUNDATIONS_CRYPTOGRAPHY_CRYPTOGRAPHY_H_
#define SRC_LIB_FOUNDATIONS_CRYPTOGRAPHY_CRYPTOGRAPHY_H_



namespace orbiter {
namespace layer1_foundations {
namespace cryptography {



// ##############################################################################
// cryptography_domain.cpp
// ##############################################################################

//! a collection of functions related to cryptography

class cryptography_domain {

public:

   cryptography_domain();
   ~cryptography_domain();
   void affine_cipher(
         std::string &ptext,
         std::string &ctext, int a, int b);
   void affine_decipher(
         std::string &ctext,
         std::string &ptext, std::string &guess);
   void vigenere_cipher(
         std::string &ptext,
         std::string &ctext, std::string &key);
   void vigenere_decipher(
         std::string &ctext,
         std::string &ptext, std::string &key);
   void vigenere_analysis(
         std::string &ctext);
   void vigenere_analysis2(
         std::string &ctext, int key_length);
```

```
int kasiski_test(
      std::string &ctext, int threshold);
void print_candidates(
      std::string &ctext,
      int i, int h, int nb_candidates, int *candidates);
void print_set(int l, int *s);
void print_on_top(
      std::string &text1,
      std::string &text2);
void decipher(
      std::string &ctext,
      std::string &ptext, std::string &guess);
void analyze(
      std::string &text);
double friedman_index(
      int *mult, int n);
double friedman_index_shifted(
      int *mult, int n, int shift);
void print_frequencies(int *mult);
void single_frequencies(
      std::string &text, int *mult);
void single_frequencies2(
      std::string &text, int stride, int n, int *mult);
void double_frequencies(
      std::string &text, int *mult);
void substition_cipher(
      std::string &ptext,
      std::string &ctext, std::string &key);
char lower_case(char c);
char upper_case(char c);
char is_alnum(char c);
void get_random_permutation(std::string &p);

void make_affine_sequence(
      int a, int c, int m, int verbose_level);
void make_2D_plot(
      int *orbit, int orbit_len, int cnt,
      int m, int a, int c, int verbose_level);
void do_random_last(int random_nb, int verbose_level);
void do_random(
      int random_nb,
      std::string &fname_csv, int verbose_level);

void do_EC_Koblitz_encoding(
      field_theory::finite_field *F,
      int EC_b, int EC_c, int EC_s,
      std::string &pt_text, std::string &EC_message,
```

```
      int verbose_level);
void do_EC_points(
      field_theory::finite_field *F, std::string &label,
      int EC_b, int EC_c, int verbose_level);
int EC_evaluate_RHS(
      field_theory::finite_field *F,
      int EC_b, int EC_c, int x);
// evaluates x^3 + bx + c
void do_EC_add(
      field_theory::finite_field *F,
      int EC_b, int EC_c,
      std::string &pt1_text,
      std::string &pt2_text, int verbose_level);
void do_EC_cyclic_subgroup(
      field_theory::finite_field *F,
      int EC_b, int EC_c,
      std::string &pt_text, int verbose_level);
void do_EC_multiple_of(
      field_theory::finite_field *F,
      int EC_b, int EC_c,
      std::string &pt_text, int n, int verbose_level);
void do_EC_discrete_log(
      field_theory::finite_field *F,
      int EC_b, int EC_c,
      std::string &base_pt_text,
      std::string &pt_text, int verbose_level);
void do_EC_baby_step_giant_step(
      field_theory::finite_field *F,
      int EC_b, int EC_c,
      std::string &EC_bsgs_G, int EC_bsgs_N,
      std::string &EC_bsgs_cipher_text,
      int verbose_level);
void do_EC_baby_step_giant_step_decode(
      field_theory::finite_field *F,
      int EC_b, int EC_c,
      std::string &EC_bsgs_A, int EC_bsgs_N,
      std::string &EC_bsgs_cipher_text_T,
      std::string &EC_bsgs_keys,
      int verbose_level);
void do_RSA_encrypt_text(
      long int RSA_d, long int RSA_m,
      int RSA_block_size,
      std::string &RSA_encrypt_text,
      int verbose_level);
void do_RSA(
      long int RSA_d,
      long int RSA_m, int RSA_block_size,
```

```
      std::string &RSA_text,
      int verbose_level);

void NTRU_encrypt(
      int N, int p,
      field_theory::finite_field *Fq,
      std::string &H_coeffs,
      std::string &R_coeffs,
      std::string &Msg_coeffs,
      int verbose_level);
void polynomial_center_lift(
      std::string &A_coeffs,
      field_theory::finite_field *F,
      int verbose_level);
void polynomial_reduce_mod_p(
      std::string &A_coeffs,
      field_theory::finite_field *F,
      int verbose_level);

void do_solovay_strassen(
      int p, int a, int verbose_level);
void do_miller_rabin(
      int p, int nb_times, int verbose_level);
void do_fermat_test(
      int p, int nb_times, int verbose_level);
void do_find_pseudoprime(
      int nb_digits, int nb_fermat,
      int nb_miller_rabin,
      int nb_solovay_strassen,
      int verbose_level);
void do_find_strong_pseudoprime(int nb_digits,
      int nb_fermat, int nb_miller_rabin,
      int verbose_level);
void do_miller_rabin_text(std::string &number_text,
      int nb_miller_rabin, int verbose_level);
void quadratic_sieve(int n, int factorbase,
      int x0, int verbose_level);
void calc_log2(std::vector<int> &primes,
      std::vector<int> &primes_log2,
      int verbose_level);
void all_square_roots_mod_n_by_exhaustive_search_lint(
      std::string &square_root_a,
      std::string &square_root_mod_n,
      std::vector<long int> &S,
      int verbose_level);
void square_root(std::string &square_root_number,
      int verbose_level);
```

```
void square_root_mod(std::string &square_root_number,
      std::string &mod_number, int verbose_level);
void reduce_primes(std::vector<int> &primes,
      ring_theory::longinteger_object &M,
      int &f_found_small_factor, int &small_factor,
      int verbose_level);
void do_sift_smooth(int sift_smooth_from,
      int sift_smooth_len,
      std::string &sift_smooth_factor_base,
      int verbose_level);
void do_discrete_log(long int y,
      long int a, long int p, int verbose_level);
void do_primitive_root(long int p, int verbose_level);
void do_primitive_root_longinteger(
      ring_theory::longinteger_object &p,
      int verbose_level);
void do_smallest_primitive_root(
      long int p, int verbose_level);
void do_smallest_primitive_root_interval(
      long int p_min, long int p_max, int verbose_level);
void do_number_of_primitive_roots_interval(
      long int p_min, long int p_max,
      int verbose_level);
void do_inverse_mod(
      long int a, long int n, int verbose_level);
void do_extended_gcd(
      int a, int b, int verbose_level);
void do_power_mod(
      ring_theory::longinteger_object &a,
      ring_theory::longinteger_object &k,
      ring_theory::longinteger_object &n,
      int verbose_level);


void calc_roots(
      ring_theory::longinteger_object &M,
      ring_theory::longinteger_object &sqrtM,
   std::vector<int> &primes,
   std::vector<int> &R1, std::vector<int> &R2,
   int verbose_level);
void Quadratic_Sieve(
   int factorbase,
   int f_mod, int mod_n, int mod_r, int x0,
   int n, ring_theory::longinteger_object &M,
   ring_theory::longinteger_object &sqrtM,
   std::vector<int> &primes,
   std::vector<int> &primes_log2,
```

```
   std::vector<int> &R1, std::vector<int> &R2,
   std::vector<int> &X,
   int verbose_level);
int quadratic_sieve(
      ring_theory::longinteger_object& M,
      ring_theory::longinteger_object& sqrtM,
   std::vector<int> &primes,
   std::vector<int> &primes_log2,
   std::vector<int> &R1, std::vector<int> &R2,
   int from, int to,
   int ll, std::vector<int> &X, int verbose_level);
int factor_over_factor_base(
      ring_theory::longinteger_object &x,
      std::vector<int> &primes,
      std::vector<int> &factor_idx,
      std::vector<int> &factor_exp,
      int verbose_level);
int factor_over_factor_base2(
      ring_theory::longinteger_object &x,
      std::vector<int> &primes,
      std::vector<int> &exponents,
      int verbose_level);

void find_probable_prime_above(
      ring_theory::longinteger_object &a,
   int nb_solovay_strassen_tests, int f_miller_rabin_test,
   int verbose_level);
int solovay_strassen_is_prime(
      ring_theory::longinteger_object &n,
      int nb_tests, int verbose_level);
int solovay_strassen_is_prime_single_test(
      ring_theory::longinteger_object &n,
      int verbose_level);
int fermat_test_iterated_with_latex_key(
      std::ostream &ost,
      ring_theory::longinteger_object &P,
      int nb_times,
      int verbose_level);
int fermat_test_with_latex_key(
      std::ostream &ost,
      ring_theory::longinteger_object &n,
      ring_theory::longinteger_object &a,
   int verbose_level);
int solovay_strassen_test(
      ring_theory::longinteger_object &n,
      ring_theory::longinteger_object &a,
   int verbose_level);
```

```
   int solovay_strassen_test_with_latex_key(
         std::ostream &ost,
         ring_theory::longinteger_object &n,
         ring_theory::longinteger_object &a,
      int verbose_level);
   int solovay_strassen_test_iterated_with_latex_key(
         std::ostream &ost,
         ring_theory::longinteger_object &P,
         int nb_times,
         int verbose_level);
   // returns TRUE is the test is conclusive,
   // i.e. if the number is not prime.
   int miller_rabin_test(
         ring_theory::longinteger_object &n,
         int verbose_level);
   int miller_rabin_test_with_latex_key(
         std::ostream &ost,
         ring_theory::longinteger_object &n,
         int iteration, int verbose_level);
   int miller_rabin_test_iterated_with_latex_key(
         std::ostream &ost,
         ring_theory::longinteger_object &P, int nb_times,
         int verbose_level);
   // returns TRUE is the test is conclusive,
   // i.e. if the number is not prime.
   void get_k_bit_random_pseudoprime(
         ring_theory::longinteger_object &n, int k,
      int nb_tests_solovay_strassen,
      int f_miller_rabin_test, int verbose_level);
   void RSA_setup(ring_theory::longinteger_object &n,
         ring_theory::longinteger_object &p,
         ring_theory::longinteger_object &q,
         ring_theory::longinteger_object &a,
         ring_theory::longinteger_object &b,
      int nb_bits,
      int nb_tests_solovay_strassen,
      int f_miller_rabin_test,
      int verbose_level);
   void do_babystep_giantstep(
         long int p, long int g, long int h,
         int f_latex, std::ostream &ost,
         int verbose_level);

};

}}}
```

```
#endif /* SRC_LIB_FOUNDATIONS_CRYPTOGRAPHY_CRYPTOGRAPHY_H_ */
```

3.7 Data Structures

```
// data_structures.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005


#ifndef ORBITER_SRC_LIB_FOUNDATIONS_DATA_STRUCTURES_DATA_STRUCTURES_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_DATA_STRUCTURES_DATA_STRUCTURES_H_



namespace orbiter {
namespace layer1_foundations {
namespace data_structures {


// ##############################################################################
// algorithms.cpp
// ##############################################################################



//! catch all class for algorithms


class algorithms {
public:

   algorithms();
   ~algorithms();
   int hashing(int hash0, int a);
   int hashing_fixed_width(int hash0, int a, int bit_length);
   void uchar_print_bitwise(std::ostream &ost, unsigned char u);
   void uchar_move(unsigned char *p, unsigned char *q, int len);
   void int_swap(int& x, int& y);
   void lint_swap(long int & x, long int & y);
   void print_pointer_hex(std::ostream &ost, void *p);
   void print_uint32_hex(std::ostream &ost, uint32_t val);
   void print_uint32_binary(std::ostream &ost, uint32_t val);
   void print_hex_digit(std::ostream &ost, int digit);
   void print_bits(std::ostream &ost, char *data, int data_size);
   void read_hex_data(std::string &str,
```

```
        char *&data, int &data_size, int verbose_level);
   unsigned char read_hex_digit(char digit);
   void print_repeated_character(std::ostream &ost, char c, int n);
   uint32_t root_of_tree_uint32_t (uint32_t* S, uint32_t i);
   void solve_diophant(int *Inc,
      int nb_rows, int nb_cols, int nb_needed,
      int f_has_Rhs, int *Rhs,
      long int *&Solutions, int &nb_sol,
      long int &nb_backtrack, int &dt,
      int f_DLX,
      int verbose_level);
   // allocates Solutions[nb_sol * nb_needed]
   uint32_t SuperFastHash (const char * data, int len);
   uint32_t SuperFastHash_uint(const unsigned int * p, int sz);
   void union_of_sets(
        std::string &fname_set_of_sets,
        std::string &fname_input,
        std::string &fname_output, int verbose_level);

};

// ##############################################################################
// bitmatrix.cpp
// ##############################################################################

//! matrices over GF(2) stored as bitvectors

class bitmatrix {
public:
   int m;
   int n;
   int N;
   uint32_t *data;

   bitmatrix();
   ~bitmatrix();
   void init(int m, int n, int verbose_level);
   void unrank_PG_elements_in_columns_consecutively(
        field_theory::finite_field *F,
        long int start_value, int verbose_level);
   void rank_PG_elements_in_columns(
        field_theory::finite_field *F,
        int *perms, unsigned int *PG_ranks,
        int verbose_level);
   void print();
   void zero_out();
   int s_ij(int i, int j);
```

```
   void m_ij(int i, int j, int a);
   void mult_int_matrix_from_the_left(
         int *A, int Am, int An,
         bitmatrix *Out, int verbose_level);

};

// ##############################################################################
// bitvector.cpp
// ##############################################################################

//! compact storage of 0/1-data as bitvectors

class bitvector {

private:
   unsigned char *data; // [allocated_length]
   long int length; // number of bits used
   long int allocated_length;


public:

   bitvector();
   ~bitvector();
   void allocate(long int length);
   void zero();
   long int get_length();
   long int get_allocated_length();
   unsigned char *get_data();
   void m_i(long int i, int a);
   void set_bit(long int i);
   int s_i(long int i);
   void save(std::ofstream &fp);
   void load(std::ifstream &fp);
   uint32_t compute_hash();
   void print();

};

// ##############################################################################
// classify_bitvectors.cpp
// ##############################################################################

//! classification of 0/1 matrices using canonical forms

class classify_bitvectors {
```

```
public:

   int nb_types;
      // the number of isomorphism types

   int rep_len;
      // the number of char we need to store the canonical form of
      // one object


   uchar **Type_data;
      // Type_data[nb_types][rep_len]
      // the canonical form of the i-th representative is
      // Type_data[i][rep_len]
   int *Type_rep;
      // Type_rep[nb_types]
      // Type_rep[i] is the index of the candidate which
      // has been chosen as representative
      // for the i-th isomorphism type
   int *Type_mult;
      // Type_mult[nb_types]
      // Type_mult[i] gives the number of candidates which
      // are isomorphic to the i-th isomorphism class representative
   void **Type_extra_data;
      // Type_extra_data[nb_types]
      // Type_extra_data[i] is a pointer that is stored with the
      // i-th isomorphism class representative

   int N;
      // number of candidates (or objects) that we will test
   int n;
      // number of candidates that we have already tested

   int *type_of;
      // type_of[nb_types]
      // type_of[i] is the isomorphism type of the i-th candidate

   tally *C_type_of;
      // the classification of type_of[nb_types]
      // this will be computed in finalize()

   int *perm;
      // the permutation which lists the orbit
      // representative in the order
      // in which they appear in the list of candidates

   classify_bitvectors();
```

```
   ~classify_bitvectors();
   void init(int N, int rep_len, int verbose_level);
   int search(uchar *data, int &idx, int verbose_level);
   void search_and_add_if_new(uchar *data,
         void *extra_data, int &f_found, int &idx,
         int verbose_level);
   int compare_at(uchar *data, int idx);
   void add_at_idx(uchar *data,
         void *extra_data, int idx, int verbose_level);
   void finalize(int verbose_level);
   void print_reps();
   void print_table();
   void save(
         std::string &prefix,
      void (*encode_function)(void *extra_data,
         long int *&encoding, int &encoding_sz, void *global_data),
      void (*get_group_order_or_NULL)(void *extra_data,
            ring_theory::longinteger_object &go, void *global_data),
      void *global_data,
      int verbose_level);

};




// ##############################################################################
// classify_using_canonical_forms.cpp
// ##############################################################################

//! classification of objects using canonical forms

class classify_using_canonical_forms {
public:


   int nb_input_objects;


   std::vector<bitvector *> B;
   std::vector<void *> Objects;
   std::vector<long int> Ago;
   std::vector<int> input_index;

   std::multimap<uint32_t, int> Hashing;
      // we store the pair (hash, idx)
      // where hash is the hash value of the set and idx is the
```

```
      // index in the table Sets where the set is stored.
      //
      // we use a multimap because the hash values are not unique
      // it happens that two sets have the same hash value.
      // map cannot handle that.


   //std::vector<void *> Input_objects;
   //std::vector<int> orbit_rep_of_input_object;

   classify_using_canonical_forms();
   ~classify_using_canonical_forms();
   void orderly_test(
         geometry::object_with_canonical_form *OwCF,
         int &f_accept, int verbose_level);
   void find_object(
         geometry::object_with_canonical_form *OwCF,
         int &f_found, int &idx,
         nauty_output *&NO,
         bitvector *&Canonical_form,
         int verbose_level);
      // if f_found is TRUE, B[idx] agrees with the given object
   void add_object(
         geometry::object_with_canonical_form *OwCF,
         int &f_new_object,
         int verbose_level);

};

// ##############################################################################
// data_file.cpp
// ##############################################################################

//! to read data files from the poset classification algorithm


class data_file {

   public:

   std::string fname;
   int nb_cases;
   int *set_sizes;
   long int **sets;
   int *casenumbers;
   char **Ago_ascii;
   char **Aut_ascii;
```

```
   int f_has_candidates;
   int *nb_candidates;
   int **candidates;

   data_file();
   ~data_file();
   void read(
         std::string &fname, int f_casenumbers,
         int verbose_level);
   void read_candidates(
         std::string &candidates_fname,
         int verbose_level);
};

// ##############################################################################
// data_input_stream_description_element.cpp:
// ##############################################################################


//! describes one element in an input stream of combinatorial objects


class data_input_stream_description_element {
public:
   enum data_input_stream_type input_type;
   std::string input_string;
   std::string input_string2;

   // for t_data_input_stream_file_of_designs:
   int input_data1; // N_points
   int input_data2; // b = number of blocks
   int input_data3; // k = block size
   int input_data4; // partition class size

   data_input_stream_description_element();
   ~data_input_stream_description_element();
   void print();
   void init_set_of_points(std::string &a);
   void init_set_of_lines(std::string &a);
   void init_set_of_points_and_lines(
         std::string &a, std::string &b);
   void init_packing(std::string &a, int q);
   void init_file_of_points(std::string &a);
   void init_file_of_lines(std::string &a);
   void init_file_of_packings(std::string &a);
   void init_file_of_packings_through_spread_table(
```

```
        std::string &a, std::string &b, int q);
   void init_file_of_point_set(std::string &a);
   void init_file_of_designs(std::string &a,
            int N_points, int b, int k, int partition_class_size);
   void init_file_of_incidence_geometries(std::string &a,
            int v, int b, int f);
   void init_file_of_incidence_geometries_by_row_ranks(
        std::string &a,
            int v, int b, int r);
   void init_incidence_geometry(std::string &a,
            int v, int b, int f);
   void init_incidence_geometry_by_row_ranks(std::string &a,
            int v, int b, int r);
   void init_from_parallel_search(std::string &fname_mask,
        int nb_cases, std::string &cases_fname);

};


// ##########################################################################
// data_input_stream_description.cpp:
// ##########################################################################


//! description of input data for classification of geometric objects from the co
mmand line


class data_input_stream_description {
public:


   int nb_inputs;

   std::vector<data_input_stream_description_element> Input;

   data_input_stream_description();
   ~data_input_stream_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();
   void print_item(int i);


};
```

```
// ##############################################################################
// data_input_stream.cpp:
// ##############################################################################


//! input data for classification of geometric objects from the command line


class data_input_stream {
public:

   data_input_stream_description *Descr;

   int nb_objects_to_test;

   std::vector<void *> Objects;

   data_input_stream();
   ~data_input_stream();
   void init(data_input_stream_description *Descr, int verbose_level);
   int count_number_of_objects_to_test(
      int verbose_level);
   void read_objects(int verbose_level);


};


// ##############################################################################
// data_structures_global.cpp:
// ##############################################################################


//! a catch-all container class for everything related to data structures


class data_structures_global {
public:
   data_structures_global();
   ~data_structures_global();
   void bitvector_m_ii(uchar *bitvec, long int i, int a);
   void bitvector_set_bit(uchar *bitvec, long int i);
   int bitvector_s_i(uchar *bitvec, long int i);
   uint32_t int_vec_hash(int *data, int len);
   uint64_t lint_vec_hash(long int *data, int len);
   uint32_t char_vec_hash(char *data, int len);
   int int_vec_hash_after_sorting(int *data, int len);
```

```
   long int lint_vec_hash_after_sorting(long int *data, int len);

};


// ##############################################################################
// fancy_set.cpp
// ##############################################################################

//! subset of size k of a set of size n


class fancy_set {

   public:

   int n;
   int k;
   long int *set;
   long int *set_inv;

   fancy_set();
   ~fancy_set();
   void init(int n, int verbose_level);
   void init_with_set(int n, int k, int *subset, int verbose_level);
   void print();
   void println();
   void swap(int pos, int a);
   int is_contained(int a);
   void copy_to(fancy_set *to);
   void add_element(int elt);
   void add_elements(int *elts, int nb);
   void delete_elements(int *elts, int nb);
   void delete_element(int elt);
   void select_subset(int *elts, int nb);
   void intersect_with(int *elts, int nb);
   void subtract_set(fancy_set *set_to_subtract);
   void sort();
   int compare_lexicographically(fancy_set *second_set);
   void complement(fancy_set *compl_set);
   int is_subset(fancy_set *set2);
   int is_equal(fancy_set *set2);
   void save(std::string &fname, int verbose_level);

};

// ##############################################################################
```

```
// int_matrix.cpp:
// ##############################################################################


//! matrices over int


class int_matrix {
public:

   int *M;
   int m;
   int n;

   int *perm_inv;
   int *perm;

   int_matrix();
   ~int_matrix();
   void null();
   void freeself();
   void allocate(int m, int n);
   void allocate_and_init(int m, int n, int *Mtx);
   int &s_ij(int i, int j);
   int &s_m();
   int &s_n();
   void print();
   void sort_rows(int verbose_level);
   void check_that_entries_are_distinct(int verbose_level);
   int search(int *entry, int &idx, int verbose_level);
   void write_csv(std::string &fname, int verbose_level);

};


// ##############################################################################
// int_vec.cpp:
// ##############################################################################


//! int arrays

class int_vec {
public:

   int_vec();
   ~int_vec();
```

```
void add(int *v1, int *v2, int *w, int len);
void add3(int *v1, int *v2, int *v3, int *w, int len);
void apply(int *from, int *through, int *to, int len);
void apply_lint(int *from, long int *through, long int *to, int len);
int is_constant_on_subset(int *v,
   int *subset, int sz, int &value);
void take_away(int *v, int &len,
      int *take_away, int nb_take_away);
   // v must be sorted
int count_number_of_nonzero_entries(int *v, int len);
int find_first_nonzero_entry(int *v, int len);
void zero(int *v, long int len);
int is_zero(int *v, long int len);
void mone(int *v, long int len);
void copy(int *from, int *to, long int len);
void copy_to_lint(int *from, long int *to, long int len);
void swap(int *v1, int *v2, long int len);
void delete_element_assume_sorted(int *v,
   int &len, int a);
void complement(int *v, int n, int k);
// computes the complement to v + k (v must be allocated to n elements)
// the first k elements of v[] must be in increasing order.
void complement(int *v, int *w, int n, int k);
// computes the complement of v[k] in the set {0,...,n-1} to w[n - k]
void init5(int *v, int a0, int a1, int a2, int a3, int a4);
int minimum(int *v, int len);
int maximum(int *v, int len);
void copy(int len, int *from, int *to);
int first_difference(int *p, int *q, int len);
int vec_max_log_of_entries(std::vector<std::vector<int> > &p);
void vec_print(std::vector<std::vector<int> > &p);
void vec_print(std::vector<std::vector<int> > &p, int w);
void distribution_compute_and_print(std::ostream &ost,
   int *v, int v_len);
void distribution(int *v,
   int len_v, int *&val, int *&mult, int &len);
void print(std::ostream &ost, std::vector<int> &v);
void print(std::ostream &ost, int *v, int len);
void print_str(std::stringstream &ost, int *v, int len);
void print_str_naked(std::stringstream &ost, int *v, int len);
void print_as_table(std::ostream &ost, int *v, int len, int width);
void print_fully(std::ostream &ost, std::vector<int> &v);
void print_fully(std::ostream &ost, int *v, int len);
void print_dense(std::ostream &ost, int *v, int len);
void print_Cpp(std::ostream &ost, int *v, int len);
void print_GAP(std::ostream &ost, int *v, int len);
void print_classified(int *v, int len);
```

```
   void print_classified_str(std::stringstream &sstr,
         int *v, int len, int f_backwards);
   void scan(std::string &s, int *&v, int &len);
   void scan_from_stream(std::istream & is, int *&v, int &len);
   void print_to_str(std::string &s, int *data, int len);
   void print_to_str_naked(std::string &s, int *data, int len);
   void print(int *v, int len);
   void print_integer_matrix(std::ostream &ost,
      int *p, int m, int n);
   void print_integer_matrix_width(std::ostream &ost,
      int *p, int m, int n, int dim_n, int w);
   void print_integer_matrix_in_C_source(std::ostream &ost,
      int *p, int m, int n);
   void matrix_make_block_matrix_2x2(int *Mtx,
      int k, int *A, int *B, int *C, int *D);
   void matrix_delete_column_in_place(int *Mtx,
      int k, int n, int pivot);
   int matrix_max_log_of_entries(int *p, int m, int n);
   void matrix_print_ost(std::ostream &ost, int *p, int m, int n);
   void matrix_print_makefile_style_ost(
         std::ostream &ost, int *p, int m, int n);
   void matrix_print(int *p, int m, int n);
   void matrix_print_tight(int *p, int m, int n);
   void matrix_print_ost(
         std::ostream &ost, int *p, int m, int n, int w);
   void matrix_print_makefile_style_ost(
         std::ostream &ost, int *p, int m, int n, int w);
   void matrix_print(int *p, int m, int n, int w);
   void matrix_print_bitwise(int *p, int m, int n);
   void distribution_print(std::ostream &ost,
      int *val, int *mult, int len);
   void distribution_print_to_string(
         std::string &str, int *val, int *mult, int len);
   void set_print(std::ostream &ost, int *v, int len);
   void integer_vec_print(std::ostream &ost, int *v, int len);
   int hash(int *v, int len, int bit_length);
   void create_string_with_quotes(
         std::string &str, int *v, int len);
   void transpose(int *M, int m, int n, int *Mt);

};



// ##############################################################################
// int_vector.cpp
// ##############################################################################
```

```
//! vector of int

class int_vector {
public:

   long int *M;
   int m;
   int alloc_length;

   int_vector();
   ~int_vector();
   void allocate(int len);
   void allocate_and_init(int len, long int *V);
   void allocate_and_init_int(int len, int *V);
   void init_permutation_from_string(std::string &s);
   void read_ascii_file(std::string &fname);
   void read_binary_file_int4(std::string &fname);
   long int &s_i(int i);
   int &length();
   void print(std::ostream &ost);
   void zero();
   int search(int a, int &idx);
   void sort();
   void make_space();
   void append(int a);
   void insert_at(int a, int idx);
   void insert_if_not_yet_there(int a);
   void sort_and_remove_duplicates();
   void write_to_ascii_file(std::string &fname);
   void write_to_binary_file_int4(std::string &fname);
   void write_to_csv_file(std::string &fname, std::string &label);
   uint32_t hash();
   int minimum();
   int maximum();



};


// ##############################################################################
// lint_vec.cpp:
// ##############################################################################


//! long int arrays
```

```
class lint_vec {
public:

   lint_vec();
   ~lint_vec();
   void apply(long int *from, long int *through, long int *to, int len);
   void take_away(long int *v, int &len,
         long int *take_away, int nb_take_away);
   void zero(long int *v, long int len);
   void mone(long int *v, long int len);
   void copy(long int *from, long int *to, long int len);
   void copy_to_int(long int *from, int *to, long int len);
   void complement(long int *v, long int *w, int n, int k);
   long int minimum(long int *v, int len);
   long int maximum(long int *v, int len);
   void matrix_print_width(std::ostream &ost,
      long int *p, int m, int n, int dim_n, int w);
   void set_print(long int *v, int len);
   void set_print(std::ostream &ost, long int *v, int len);
   void print(std::ostream &ost, long int *v, int len);
   void print(std::ostream &ost, std::vector<long int> &v);
   void print_as_table(std::ostream &ost, long int *v, int len, int width);
   void print_bare_fully(std::ostream &ost, long int *v, int len);
   void print_fully(std::ostream &ost, long int *v, int len);
   void print_fully(std::ostream &ost, std::vector<long int> &v);
   int matrix_max_log_of_entries(long int *p, int m, int n);
   void matrix_print(long int *p, int m, int n);
   void matrix_print(long int *p, int m, int n, int w);
   void scan(std::string &s, long int *&v, int &len);
   void scan_from_stream(std::istream & is, long int *&v, int &len);
   void print_to_str(std::string &s, long int *data, int len);
   void print_to_str_naked(std::string &s, long int *data, int len);
   void create_string_with_quotes(std::string &str, long int *v, int len);


};



// ##############################################################################
// nauty_output.cpp:
// ##############################################################################


//! output data created by a run of nauty
```

```
class nauty_output {
public:


   int N;

   int *Aut;  // [Aut_counter * N]
   int Aut_counter;

   int *Base; // [Base_length]
   int Base_length;

   long int *Base_lint;
   int *Transversal_length;


   ring_theory::longinteger_object *Ago;

   int *canonical_labeling; // [N]

   long int nb_firstpathnode;
   long int nb_othernode;
   long int nb_processnode;
   long int nb_firstterminal;

   nauty_output();
   ~nauty_output();
   void allocate(int N, int verbose_level);
   void print();
   void print_stats();
   int belong_to_the_same_orbit(int a, int b, int verbose_level);

};


// ##############################################################################
// page_storage.cpp
// ##############################################################################

//! bulk storage of group elements in compressed form

class page_storage {

public:
   long int overall_length;

   long int entry_size; // in char
```

```
   long int page_length_log; // number of bits
   long int page_length; // entries per page
   long int page_size; // size in char of one page
   long int allocation_table_length;
      // size in char of one allocation table

   long int page_ptr_used;
   long int page_ptr_allocated;
   long int page_ptr_oversize;

   uchar **pages;
   uchar **allocation_tables;

   long int next_free_entry;
   long int nb_free_entries;

   int f_elt_print_function;
   void (* elt_print)(void *p, void *data, std::ostream &ost);
   void *elt_print_data;


   void init(int entry_size, int page_length_log,
      int verbose_level);
   void add_elt_print_function(
      void (* elt_print)(void *p, void *data, std::ostream &ost),
      void *elt_print_data);
   void print();
   uchar *s_i_and_allocate(long int i);
   uchar *s_i_and_deallocate(long int i);
   uchar *s_i(long int i);
   uchar *s_i_and_allocation_bit(long int i, int &f_allocated);
   void check_allocation_table();
   long int store(uchar *elt);
   void dispose(long int hdl);
   void check_free_list();
   page_storage();
   ~page_storage();
   void print_storage_used();

};



// ##############################################################################
// partitionstack.cpp
// ##############################################################################
```

```
//! data structure for set partitions following Jeffrey Leon


class partitionstack {
   public:

   // data structure for the partition stack,
   // following Leon:
      int n; // size of the set that is partitioned
      int ht;
      int ht0;

      int *pointList, *invPointList;
      int *cellNumber;

      int *startCell;
      int *cellSize;
      int *parent;


   // for matrix canonization:
   // int first_column_element;

   // subset to be chosen by classify_by_..._extract_subset():
   // used as input for split_cell()
      //
   // used if SPLIT_MULTIPLY is defined:
      int nb_subsets;
      int *subset_first;
      int *subset_length;
      int *subsets;
      //
   // used if SPLIT_MULTIPLY is not defined:
      int *subset;
      int subset_size;

   partitionstack();
   ~partitionstack();
   void free();
   void allocate(int n, int verbose_level);
   void allocate_with_two_classes(
         int n, int v, int b, int verbose_level);
   int parent_at_height(int h, int cell);
   int is_discrete();
   int smallest_non_discrete_cell();
```

```
int biggest_non_discrete_cell();
int smallest_non_discrete_cell_rows_preferred();
int biggest_non_discrete_cell_rows_preferred();
int nb_partition_classes(int from, int len);
int is_subset_of_cell(int *set, int size, int &cell_idx);
void sort_cells();
void sort_cell(int cell);
void reverse_cell(int cell);
void check();
void print_raw();
void print_class(std::ostream& ost, int idx);
void print_classes_tex(std::ostream& ost);
void print_class_tex(std::ostream& ost, int idx);
void print_class_point_or_line(std::ostream& ost, int idx);
void print_classes(std::ostream& ost);
void print_classes_points_and_lines(std::ostream& ost);
std::ostream& print(std::ostream& ost);
void print_cell(int i);
void print_cell_latex(std::ostream &ost, int i);
void print_subset();
void get_cell(int i,
      int *&cell, int &cell_sz, int verbose_level);
void get_cell_lint(int i,
      long int *&cell, int &cell_sz,
      int verbose_level);
void get_row_classes(
      set_of_sets *&Sos, int verbose_level);
void get_column_classes(
      set_of_sets *&Sos, int verbose_level);
void write_cell_to_file(int i,
      std::string &fname, int verbose_level);
void write_cell_to_file_points_or_lines(int i,
      std::string &fname, int verbose_level);
void refine_arbitrary_set_lint(
      int size, long int *set, int verbose_level);
void refine_arbitrary_set(
      int size, int *set, int verbose_level);
void split_cell(int verbose_level);
void split_multiple_cells(int *set, int set_size,
   int f_front, int verbose_level);
void split_line_cell_front_or_back(int *set, int set_size,
   int f_front, int verbose_level);
void split_cell_front_or_back(int *set, int set_size,
   int f_front, int verbose_level);
void split_cell(int *set, int set_size, int verbose_level);
void join_cell();
void reduce_height(int ht0);
```

```
void isolate_point(int pt);
void subset_continguous(int from, int len);
int is_row_class(int c);
int is_col_class(int c);
void allocate_and_get_decomposition(
   int *&row_classes, int *&row_class_inv,
      int &nb_row_classes,
   int *&col_classes, int *&col_class_inv,
      int &nb_col_classes,
   int verbose_level);
void get_row_and_col_permutation(
   int *row_classes, int nb_row_classes,
   int *col_classes, int nb_col_classes,
   int *row_perm, int *row_perm_inv,
   int *col_perm, int *col_perm_inv);
void get_row_and_col_classes(int *row_classes,
   int &nb_row_classes,
   int *col_classes, int &nb_col_classes,
   int verbose_level);
void initial_matrix_decomposition(int nbrows,
   int nbcols,
   int *V, int nb_V, int *B, int nb_B,
   int verbose_level);
int is_descendant_of(int cell, int ancestor_cell,
   int verbose_level);
int is_descendant_of_at_level(int cell, int ancestor_cell,
      int level, int verbose_level);
int cellSizeAtLevel(int cell, int level);

void print_decomposition_tex(std::ostream &ost,
   int *row_classes, int nb_row_classes,
   int *col_classes, int nb_col_classes);
void print_decomposition_scheme(std::ostream &ost,
   int *row_classes, int nb_row_classes,
   int *col_classes, int nb_col_classes,
   int *scheme, int marker1, int marker2);
void print_decomposition_scheme_tex(std::ostream &ost,
   int *row_classes, int nb_row_classes,
   int *col_classes, int nb_col_classes,
   int *scheme);
void print_tactical_decomposition_scheme_tex_internal(
      std::ostream &ost, int f_enter_math_mode,
   int *row_classes, int nb_row_classes,
   int *col_classes, int nb_col_classes,
   int *row_scheme, int *col_scheme,
   int f_print_subscripts);
void print_tactical_decomposition_scheme_tex(
```

```
        std::ostream &ost,
     int *row_classes, int nb_row_classes,
     int *col_classes, int nb_col_classes,
     int *row_scheme, int *col_scheme,
     int f_print_subscripts);
  void print_row_tactical_decomposition_scheme_tex(
        std::ostream &ost, int f_enter_math_mode,
     int *row_classes, int nb_row_classes,
     int *col_classes, int nb_col_classes,
     int *row_scheme, int f_print_subscripts);
  void print_column_tactical_decomposition_scheme_tex(
        std::ostream &ost, int f_enter_math_mode,
     int *row_classes, int nb_row_classes,
     int *col_classes, int nb_col_classes,
     int *col_scheme, int f_print_subscripts);
  void print_non_tactical_decomposition_scheme_tex(
        std::ostream &ost, int f_enter_math_mode,
     int *row_classes, int nb_row_classes,
     int *col_classes, int nb_col_classes,
     int f_print_subscripts);
  int hash_column_refinement_info(
        int ht0, int *data, int depth,
     int hash0);
  int hash_row_refinement_info(
        int ht0, int *data, int depth, int hash0);
  void print_column_refinement_info(
        int ht0, int *data, int depth);
  void print_row_refinement_info(
        int ht0, int *data, int depth);
  void radix_sort(
        int left, int right, int *C,
     int length, int radix, int verbose_level);
  void radix_sort_bits(
        int left, int right,
     int *C, int length, int radix, int mask,
     int verbose_level);
  void swap_ij(int *perm, int *perm_inv, int i, int j);
  int my_log2(int m);
  void split_by_orbit_partition(
        int nb_orbits,
     int *orbit_first, int *orbit_len, int *orbit,
     int offset,
     int verbose_level);
};

// ##############################################################################
// set_builder_description.cpp
```

```
// ##############################################################################



//! to define a set of integers for class set_builder


class set_builder_description {
public:

   int f_index_set_loop;
   int index_set_loop_low;
   int index_set_loop_upper_bound;
   int index_set_loop_increment;

   int f_affine_function;
   int affine_function_a;
   int affine_function_b;

   int f_clone_with_affine_function;
   int clone_with_affine_function_a;
   int clone_with_affine_function_b;

   int f_set_builder;
   set_builder_description *Descr;

   int f_here;
   std::string here_text;

   int f_file;
   std::string file_name;

   int f_file_orbiter_format;
   std::string file_orbiter_format_name;

   set_builder_description();
   ~set_builder_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();
};



// ##############################################################################
// set_builder.cpp
```

```
// ##############################################################################



//! to create a set of integers from class set_builder_description


class set_builder {
public:

   set_builder_description *Descr;

   long int *set;
   int sz;

   set_builder();
   ~set_builder();
   void init(set_builder_description *Descr, int verbose_level);
   long int process_transformations(long int x);
   long int clone_with_affine_function(long int x);
};


// ##############################################################################
// set_of_sets_lint.cpp
// ##############################################################################

//! set of sets with entries over long int


class set_of_sets_lint {

public:

   long int underlying_set_size;
   int nb_sets;
   long int **Sets;
   int *Set_size;


   set_of_sets_lint();
   ~set_of_sets_lint();
   void init_simple(long int underlying_set_size,
         int nb_sets, int verbose_level);
   void init(long int underlying_set_size,
         int nb_sets, long int **Pts, int *Sz, int verbose_level);
   void init_basic(long int underlying_set_size,
```

```
         int nb_sets, int *Sz, int verbose_level);
   void init_set(int idx_of_set,
         long int *set, int sz, int verbose_level);
};




// ##############################################################################
// set_of_sets.cpp
// ##############################################################################

//! set of sets


class set_of_sets {

public:

   int underlying_set_size;
   int nb_sets;
   long int **Sets;
   long int *Set_size;


   set_of_sets();
   ~set_of_sets();
   set_of_sets *copy();
   void init_simple(int underlying_set_size,
      int nb_sets, int verbose_level);
   void init_from_adjacency_matrix(int n, int *Adj,
      int verbose_level);
   void init(int underlying_set_size, int nb_sets,
      long int **Pts, long int *Sz, int verbose_level);
   void init_with_Sz_in_int(int underlying_set_size,
         int nb_sets, long int **Pts, int *Sz,
         int verbose_level);
   void init_basic(int underlying_set_size,
      int nb_sets, long int *Sz, int verbose_level);
   void init_basic_with_Sz_in_int(int underlying_set_size,
         int nb_sets, int *Sz, int verbose_level);
   void init_basic_constant_size(int underlying_set_size,
      int nb_sets, int constant_size, int verbose_level);
   void init_from_file(int &underlying_set_size,
         std::string &fname, int verbose_level);
   void init_from_csv_file(int underlying_set_size,
         std::string &fname, int verbose_level);
```

```
void init_from_orbiter_file(int underlying_set_size,
      std::string &fname, int verbose_level);
void init_set(int idx_of_set, int *set, int sz,
   int verbose_level);
   // Stores a copy of the given set.
void init_cycle_structure(
      int *perm, int n, int verbose_level);
int total_size();
long int &element(int i, int j);
void add_element(int i, long int a);
void print();
void print_table();
void print_table_tex(std::ostream &ost);
void print_table_latex_simple(std::ostream &ost);
void print_table_latex_simple_with_selection(
      std::ostream &ost, int *Selection, int nb_sel);
void dualize(set_of_sets *&S, int verbose_level);
void remove_sets_of_given_size(int k,
   set_of_sets &S, int *&Idx,
   int verbose_level);
void extract_largest_sets(set_of_sets &S,
   int *&Idx, int verbose_level);
void intersection_matrix(
   int *&intersection_type, int &highest_intersection_number,
   int *&intersection_matrix, int &nb_big_sets,
   int verbose_level);
void compute_incidence_matrix(int *&Inc, int &m, int &n,
   int verbose_level);
void compute_and_print_tdo_row_scheme(std::ostream &file,
   int verbose_level);
void compute_and_print_tdo_col_scheme(std::ostream &file,
   int verbose_level);
void init_decomposition(
      geometry::decomposition *&D, int verbose_level);
void compute_tdo_decomposition(
      geometry::decomposition &D,
   int verbose_level);
int is_member(int i, int a, int verbose_level);
void sort_all(int verbose_level);
void all_pairwise_intersections(set_of_sets *&Intersections,
   int verbose_level);
void pairwise_intersection_matrix(
      int *&M, int verbose_level);
void all_triple_intersections(set_of_sets *&Intersections,
   int verbose_level);
int has_constant_size_property();
int largest_set_size();
```

```
   void save_csv(std::string &fname,
      int f_make_heading, int verbose_level);
   void save_constant_size_csv(std::string &fname,
         int verbose_level);
   int find_common_element_in_two_sets(int idx1, int idx2,
      int &common_elt);
   void sort();
   void sort_big(int verbose_level);
   void compute_orbits(int &nb_orbits,
         int *&orbit, int *&orbit_inv,
      int *&orbit_first, int *&orbit_len,
      void (*compute_image_function)(set_of_sets *S,
         void *compute_image_data, int elt_idx, int gen_idx,
         int &idx_of_image, int verbose_level),
      void *compute_image_data,
      int nb_gens,
      int verbose_level);
   int number_of_eckardt_points(int verbose_level);
   void get_eckardt_points(
         int *&E, int &nb_E, int verbose_level);
   void evaluate_function_and_store(
         data_structures::set_of_sets *&Function_values,
         int (*evaluate_function)(int a, int i, int j,
               void *evaluate_data, int verbose_level),
         void *evaluate_data,
         int verbose_level);
   int find_smallest_class();
};



// ##############################################################################
// sorting.cpp
// ##############################################################################

//! a collection of functions related to sorted vectors


class sorting {

public:
   sorting();
   ~sorting();

   void int_vec_search_vec(
         int *v, int len, int *A, int A_sz, int *Idx);
   void lint_vec_search_vec(
```

```
      long int *v, int len,
      long int *A, int A_sz, long int *Idx);
void int_vec_search_vec_linear(
      int *v, int len, int *A, int A_sz, int *Idx);
void lint_vec_search_vec_linear(
      long int *v, int len,
      long int *A, int A_sz, long int *Idx);
int int_vec_is_subset_of(
      int *set, int sz, int *big_set, int big_set_sz);
int lint_vec_is_subset_of(
      int *set, int sz,
      long int *big_set, int big_set_sz, int verbose_level);
void int_vec_swap_points(
      int *list, int *list_inv, int idx1, int idx2);
int int_vec_is_sorted(int *v, int len);
void int_vec_sort_and_remove_duplicates(
      int *v, int &len);
void lint_vec_sort_and_remove_duplicates(
      long int *v, int &len);
int int_vec_sort_and_test_if_contained(
      int *v1, int len1, int *v2, int len2);
int lint_vec_sort_and_test_if_contained(
      long int *v1, int len1, long int *v2, int len2);
int int_vecs_are_disjoint(int *v1, int len1, int *v2, int len2);
int int_vecs_find_common_element(int *v1, int len1,
   int *v2, int len2, int &idx1, int &idx2);
int lint_vecs_find_common_element(long int *v1, int len1,
   long int *v2, int len2, int &idx1, int &idx2);
void int_vec_insert_and_reallocate_if_necessary(
      int *&vec,
   int &used_length, int &alloc_length, int a,
   int verbose_level);
void int_vec_append_and_reallocate_if_necessary(
      int *&vec,
   int &used_length, int &alloc_length, int a,
   int verbose_level);
void lint_vec_append_and_reallocate_if_necessary(
      long int *&vec,
      int &used_length, int &alloc_length, long int a,
      int verbose_level);
int int_vec_is_zero(int *v, int len);
int test_if_sets_are_equal(
      int *set1, int *set2, int set_size);
int test_if_sets_are_disjoint(
      long int *set1, int sz1, long int *set2, int sz2);
void test_if_set(int *set, int set_size);
int lint_vec_test_if_set(long int *set, int set_size);
```

```
int test_if_set_with_return_value(
      int *set, int set_size);
int test_if_set_with_return_value_lint(
      long int *set, int set_size);
void rearrange_subset(int n, int k, int *set,
   int *subset, int *rearranged_set, int verbose_level);
void rearrange_subset_lint(int n, int k,
   long int *set, int *subset, long int *rearranged_set,
   int verbose_level);
void rearrange_subset_lint_all(int n, int k,
   long int *set, long int *subset, long int *rearranged_set,
   int verbose_level);
int int_vec_search_linear(int *v, int len, int a, int &idx);
int lint_vec_search_linear(
      long int *v, int len, long int a, int &idx);
void int_vec_intersect(int *v1, int len1, int *v2, int len2,
   int *&v3, int &len3);
void vec_intersect(long int *v1, int len1,
   long int *v2, int len2, long int *&v3, int &len3);
void int_vec_intersect_sorted_vectors(int *v1, int len1,
   int *v2, int len2, int *v3, int &len3);
void lint_vec_intersect_sorted_vectors(long int *v1, int len1,
      long int *v2, int len2, long int *v3, int &len3);
void int_vec_sorting_permutation(int *v, int len, int *perm,
   int *perm_inv, int f_increasingly);
void lint_vec_sorting_permutation(long int *v, int len,
   int *perm, int *perm_inv, int f_increasingly);
// perm and perm_inv must be allocated to len elements
void int_vec_quicksort(int *v, int (*compare_func)(int a, int b),
   int left, int right);
void lint_vec_quicksort(long int *v,
   int (*compare_func)(long int a, long int b), int left, int right);
void int_vec_quicksort_increasingly(
      int *v, int len);
void int_vec_quicksort_decreasingly(
      int *v, int len);
void lint_vec_quicksort_increasingly(
      long int *v, int len);
void lint_vec_quicksort_decreasingly(
      long int *v, int len);
void quicksort_array(int len, void **v,
   int (*compare_func)(void *a, void *b, void *data), void *data);
void quicksort_array_with_perm(
      int len, void **v, int *perm,
   int (*compare_func)(void *a, void *b, void *data), void *data);
int vec_search(
      void **v, int (*compare_func)(void *a, void *b, void *data),
```

```
   void *data_for_compare,
   int len, void *a, int &idx, int verbose_level);
int vec_search_general(void *vec,
   int (*compare_func)(void *vec,
         void *a, int b, void *data_for_compare),
   void *data_for_compare,
   int len, void *a, int &idx, int verbose_level);
int int_vec_search_and_insert_if_necessary(
      int *v, int &len, int a);
int int_vec_search_and_remove_if_found(
      int *v, int &len, int a);
int int_vec_search(int *v, int len, int a, int &idx);
   // This function finds the last occurrence of the element a.
   // If a is not found, it returns in idx the position
   // where it should be inserted if
   // the vector is assumed to be in increasing order.
int lint_vec_search(long int *v, int len, long int a,
      int &idx, int verbose_level);
   // This function finds the last occurrence of the element a.
   // If a is not found, it returns in idx the position
   // where it should be inserted if
   // the vector is assumed to be in increasing order.
int vector_lint_search(std::vector<long int> &v,
      long int a, int &idx, int verbose_level);
int int_vec_search_first_occurrence(
      int *v, int len, int a, int &idx,
      int verbose_level);
   // This function finds the first occurrence of the element a.
int lint_vec_search_first_occurrence(long int *v,
      int len, long int a, int &idx,
      int verbose_level);
// This function finds the first occurrence of the element a.
int longinteger_vec_search(
      ring_theory::longinteger_object *v, int len,
      ring_theory::longinteger_object &a, int &idx);
void int_vec_classify_and_print(
      std::ostream &ost, int *v, int l);
void int_vec_values(
      int *v, int l, int *&w, int &w_len);
void int_vec_multiplicities(
      int *v, int l, int *&w, int &w_len);
void int_vec_values_and_multiplicities(int *v, int l,
   int *&val, int *&mult, int &nb_values);
void int_vec_classify(
      int length, int *the_vec, int *&the_vec_sorted,
   int *&sorting_perm, int *&sorting_perm_inv,
   int &nb_types, int *&type_first, int *&type_len);
```

```
void int_vec_classify_with_arrays(
      int length,
   int *the_vec, int *the_vec_sorted,
   int *sorting_perm, int *sorting_perm_inv,
   int &nb_types, int *type_first, int *type_len);
void int_vec_sorted_collect_types(
      int length, int *the_vec_sorted,
   int &nb_types, int *type_first, int *type_len);
void lint_vec_sorted_collect_types(
      int length,
   long int *the_vec_sorted,
   int &nb_types, int *type_first, int *type_len);
void int_vec_print_classified(
      std::ostream &ost, int *vec, int len);
void int_vec_print_types(
      std::ostream &ost,
   int f_backwards, int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void lint_vec_print_types(std::ostream &ost,
   int f_backwards, long int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void int_vec_print_types_naked_stringstream(
      std::stringstream &sstr,
   int f_backwards, int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void lint_vec_print_types_naked_stringstream(
      std::stringstream &sstr,
   int f_backwards, long int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void int_vec_print_types_naked(
      std::ostream &ost, int f_backwards,
   int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void lint_vec_print_types_naked(
      std::ostream &ost,
   int f_backwards, long int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void int_vec_print_types_naked_tex(
      std::ostream &ost, int f_backwards,
   int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void lint_vec_print_types_naked_tex(std::ostream &ost,
   int f_backwards, long int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void int_vec_print_types_naked_tex_we_are_in_math_mode(
      std::ostream &ost,
   int f_backwards, int *the_vec_sorted,
```

```
   int nb_types, int *type_first, int *type_len);
void lint_vec_print_types_naked_tex_we_are_in_math_mode(
      std::ostream &ost,
   int f_backwards, long int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void Heapsort(void *v, int len, int entry_size_in_chars,
   int (*compare_func)(void *v1, void *v2));
void Heapsort_general(void *data, int len,
   int (*compare_func)(void *data, int i, int j, void *extra_data),
   void (*swap_func)(void *data, int i, int j, void *extra_data),
   void *extra_data);
void Heapsort_general_with_log(void *data, int *w, int len,
   int (*compare_func)(void *data,
         int i, int j, void *extra_data),
   void (*swap_func)(void *data,
         int i, int j, void *extra_data),
   void *extra_data);
int search_general(
      void *data, int len, void *search_object, int &idx,
   int (*compare_func)(void *data, int i, void *search_object,
   void *extra_data),
   void *extra_data, int verbose_level);
   // This function finds the last occurrence of the element a.
   // If a is not found, it returns in idx the position
   // where it should be inserted if
   // the vector is assumed to be in increasing order.
void int_vec_heapsort(int *v, int len);
void lint_vec_heapsort(long int *v, int len);
void int_vec_heapsort_with_log(
      int *v, int *w, int len);
void lint_vec_heapsort_with_log(
      long int *v, long int *w, int len);
void heapsort_make_heap(int *v, int len);
void lint_heapsort_make_heap(long int *v, int len);
void heapsort_make_heap_with_log(
      int *v, int *w, int len);
void lint_heapsort_make_heap_with_log(
      long int *v, long int *w, int len);
void Heapsort_make_heap(
      void *v, int len, int entry_size_in_chars,
   int (*compare_func)(void *v1, void *v2));
void Heapsort_general_make_heap(
      void *data, int len,
   int (*compare_func)(void *data, int i, int j, void *extra_data),
   void (*swap_func)(void *data, int i, int j, void *extra_data),
   void *extra_data);
void Heapsort_general_make_heap_with_log(
```

```
      void *data, int *w, int len,
   int (*compare_func)(void *data, int i, int j, void *extra_data),
   void (*swap_func)(void *data, int i, int j, void *extra_data),
   void *extra_data);
void heapsort_sift_down(
      int *v, int start, int end);
void lint_heapsort_sift_down(
      long int *v, int start, int end);
void heapsort_sift_down_with_log(
      int *v, int *w, int start, int end);
void lint_heapsort_sift_down_with_log(
      long int *v, long int *w, int start, int end);
void Heapsort_sift_down(
      void *v, int start, int end, int entry_size_in_chars,
   int (*compare_func)(void *v1, void *v2));
void Heapsort_general_sift_down(
      void *data, int start, int end,
   int (*compare_func)(void *data, int i, int j, void *extra_data),
   void (*swap_func)(void *data, int i, int j, void *extra_data),
   void *extra_data);
void Heapsort_general_sift_down_with_log(
      void *data, int *w, int start, int end,
   int (*compare_func)(void *data, int i, int j, void *extra_data),
   void (*swap_func)(void *data, int i, int j, void *extra_data),
   void *extra_data);
void heapsort_swap(int *v, int i, int j);
void lint_heapsort_swap(long int *v, int i, int j);
void Heapsort_swap(
      void *v, int i, int j, int entry_size_in_chars);
void find_points_by_multiplicity(
      int *data, int data_sz, int multiplicity,
   int *&pts, int &nb_pts);
void int_vec_bubblesort_increasing(int len, int *p);
int integer_vec_compare(int *p, int *q, int len);
int integer_vec_std_compare(
      const std::vector<unsigned int> &p,
      const std::vector<unsigned int> &q);
int lint_vec_compare(
      long int *p, long int *q, int len);
void schreier_vector_compute_depth_and_ancestor(
   int n, int *pts, int *prev, int f_use_pts_inv, int *pts_inv,
   int *&depth, int *&ancestor, int verbose_level);
int schreier_vector_determine_depth_recursion(
   int n, int *pts, int *prev, int f_use_pts_inv, int *pts_inv,
   int *depth, int *ancestor, int pos);
void schreier_vector_tree(
   int n, int *pts, int *prev, int f_use_pts_inv, int *pts_inv,
```

```
      std::string &fname_base,
      graphics::layered_graph_draw_options *LG_Draw_options,
      graph_theory::layered_graph *&LG,
      int verbose_level);
   int compare_sets(int *set1, int *set2, int sz1, int sz2);
   int compare_sets_lint(
         long int *set1, long int *set2, int sz1, int sz2);
   int test_if_sets_are_disjoint(
         long int *set1, long int *set2, int sz1, int sz2);
   void d_partition(
         double *v, int left, int right, int *middle);
   void d_quicksort(
         double *v, int left, int right);
   void d_quicksort_array(int len, double *v);
   int test_if_sets_are_disjoint_assuming_sorted(
         int *set1, int *set2, int sz1, int sz2);
   int test_if_sets_are_disjoint_assuming_sorted_lint(
         long int *set1, long int *set2, int sz1, int sz2);
   int uchar_vec_compare(uchar *p, uchar *q, int len);
   int test_if_sets_are_disjoint_not_assuming_sorted(
         long int *v, long int *w, int len);
   int int_vec_compare(int *p, int *q, int len);
   int uint_vec_compare(
         unsigned int *p, unsigned int *q, int len);
   int int_vec_compare_stride(
         int *p, int *q, int len, int stride);
   void sorted_vec_get_first_and_length(
         int *v, int len,
         int *class_first, int *class_len, int &nb_classes);
   // we assume that the vector v is sorted.

};


// ##############################################################################
// spreadsheet.cpp
// ##############################################################################

//! for reading and writing of csv files


class spreadsheet {

public:

   char **tokens;
   int nb_tokens;
```

```
int *line_start, *line_size;
int nb_lines;

int nb_rows, nb_cols;
int *Table;


spreadsheet();
~spreadsheet();
void init_set_of_sets(set_of_sets *S, int f_make_heading);
void init_int_matrix(int nb_rows, int nb_cols, int *A);
void init_empty_table(int nb_rows, int nb_cols);
void fill_entry_with_text(int row_idx,
   int col_idx, const char *text);
void fill_entry_with_text(int row_idx,
      int col_idx, std::string &text);
void set_entry_lint(int row_idx,
      int col_idx, long int val);
void fill_column_with_text(int col_idx, std::string *text,
   const char *heading);
void fill_column_with_int(int col_idx, int *data,
   const char *heading);
void fill_column_with_lint(int col_idx,
      long int *data, const char *heading);
void fill_column_with_row_index(int col_idx,
   const char *heading);
void add_token(const char *label);
void save(std::string &fname, int verbose_level);
void read_spreadsheet(std::string &fname, int verbose_level);
void print_table(std::ostream &ost, int f_enclose_in_parentheses);
void print_table_latex_all_columns(std::ostream &ost,
   int f_enclose_in_parentheses);
void print_table_latex(std::ostream &ost,
      int *f_column_select,
      int f_enclose_in_parentheses,
      int nb_lines_per_table);
void print_table_row(int row, int f_enclose_in_parentheses,
      std::ostream &ost);
void print_table_row_latex(int row, int *f_column_select,
   int f_enclose_in_parentheses, std::ostream &ost);
void print_table_row_detailed(int row, std::ostream &ost);
void print_table_row_with_column_selection(int row,
      int f_enclose_in_parentheses,
      int *Col_selection, int nb_cols_selected,
      std::ostream &ost, int verbose_level);
void print_table_with_row_selection(int *f_selected,
```

```
        std::ostream &ost);
   void print_table_sorted(std::ostream &ost, const char *sort_by);
   void add_column_with_constant_value(
        const char *label, char *value);
   void add_column_with_int(
        const char *label, int *Value);
   void add_column_with_text(
        const char *label, char **Value);
   void reallocate_table();
   void reallocate_table_add_row();
   int find_column(std::string &column_label);
   int find_by_column(const char *join_by);
   void tokenize(std::string &fname,
      char **&tokens, int &nb_tokens, int verbose_level);
   void remove_quotes(int verbose_level);
   void remove_rows(const char *drop_column, const char *drop_label,
      int verbose_level);
   void remove_rows_where_field_is_empty(
        const char *drop_column,
      int verbose_level);
   void find_rows(int verbose_level);
   void get_value_double_or_NA(int i, int j, double &val, int &f_NA);
   void get_string(std::string &str, int i, int j);
   long int get_lint(int i, int j);
   double get_double(int i, int j);
   void join_with(spreadsheet *S2, int by1, int by2,
      int verbose_level);
   void patch_with(spreadsheet *S2, char *join_by);


};


// ##############################################################################
// string_tools.cpp
// ##############################################################################

//! functions related to strings and character arrays


class string_tools {

public:

   string_tools();
   ~string_tools();
   int is_csv_file(const char *fname);
```

```
int is_inc_file(const char *fname);
int is_xml_file(const char *fname);
int s_scan_int(char **s, int *i);
int s_scan_lint(char **s, long int *i);
int s_scan_double(char **s, double *d);
int s_scan_token(char **s, char *str);
int s_scan_token_arbitrary(char **s, char *str);
int s_scan_str(char **s, char *str);
int s_scan_token_comma_separated(
      const char **s, char *str, int verbose_level);
void scan_permutation_from_string(std::string &s,
   int *&perm, int &degree, int verbose_level);
void scan_permutation_from_stream(std::istream & is,
   int *&perm, int &degree, int verbose_level);
void chop_string(const char *str, int &argc, char **&argv);
void chop_string_comma_separated(
      const char *str, int &argc, char **&argv);
void convert_arguments(
      int &argc, const char **argv, std::string *&Argv);
char get_character(std::istream & is, int verbose_level);
void replace_extension_with(char *p, const char *new_ext);
void replace_extension_with(std::string &p, const char *new_ext);
void chop_off_extension(char *p);
void chop_off_extension_and_path(std::string &p);
void chop_off_extension(std::string &p);
void chop_off_path(std::string &p);
void chop_off_extension_if_present(
      std::string &p, const char *ext);
void chop_off_extension_if_present(
      char *p, const char *ext);
void get_fname_base(const char *p, char *fname_base);
void get_extension(std::string &p, std::string &ext);
void get_extension_if_present(const char *p, char *ext);
void get_extension_if_present_and_chop_off(char *p, char *ext);
void string_fix_escape_characters(std::string &str);
void remove_specific_character(std::string &str, char c);
void create_comma_separated_list(
      std::string &output, long int *input, int input_sz);
int is_all_whitespace(const char *str);
void text_to_three_double(std::string &text, double *d);
int strcmp_with_or_without(char *p, char *q);
int starts_with_a_number(std::string &str);
int stringcmp(std::string &str, const char *p);
int strtoi(std::string &str);
int str2int(std::string &str);
long int strtolint(std::string &str);
double strtof(std::string &str);
```

```
   void parse_value_pairs(
         std::map<std::string, std::string> &symbol_table,
         std::string &evaluate_text, int verbose_level);
   void parse_comma_separated_values(
         std::vector<std::string> &symbol_table,
         std::string &evaluate_text, int verbose_level);
   void drop_quotes(std::string &in, std::string &out);
   void parse_comma_separated_strings(
         std::string &in, std::vector<std::string> &out);
   int read_schlaefli_label(const char *p);
   void read_string_of_schlaefli_labels(
         std::string &str, int *&v, int &sz, int verbose_level);


};

int string_tools_compare_strings(void *a, void *b, void *data);


// ##############################################################################
// tally_lint.cpp
// ##############################################################################


//! a statistical analysis of data consisting of long integers



class tally_lint {

public:

   int data_length;

   int f_data_ownership;
   long int *data;
   long int *data_sorted;
   int *sorting_perm;
      // computed using int_vec_sorting_permutation
   int *sorting_perm_inv;
      // perm_inv[i] is the index in data
      // of the element in data_sorted[i]
   int nb_types;
   int *type_first;
   int *type_len;

   int f_second;
```

```
int *second_data_sorted;
int *second_sorting_perm;
int *second_sorting_perm_inv;
int second_nb_types;
int *second_type_first;
int *second_type_len;

tally_lint();
~tally_lint();
void init(long int *data, int data_length,
   int f_second, int verbose_level);
void init_vector_lint(std::vector<long int> &data,
      int f_second, int verbose_level);
void sort_and_classify();
void sort_and_classify_second();
int class_of(int pt_idx);
void print(int f_backwards);
void print_no_lf(int f_backwards);
void print_tex_no_lf(int f_backwards);
void print_first(int f_backwards);
void print_second(int f_backwards);
void print_first_tex(int f_backwards);
void print_second_tex(int f_backwards);
void print_file(std::ostream &ost, int f_backwards);
void print_file_tex(std::ostream &ost, int f_backwards);
void print_file_tex_we_are_in_math_mode(
      std::ostream &ost, int f_backwards);
void print_naked_stringstream(
      std::stringstream &sstr, int f_backwards);
void print_naked(int f_backwards);
void print_naked_tex(std::ostream &ost, int f_backwards);
void print_types_naked_tex(std::ostream &ost, int f_backwards,
   int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void print_lint_types_naked_tex(
   std::ostream &ost, int f_backwards, long int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void print_array_tex(std::ostream &ost, int f_backwards);
double average();
double average_of_non_zero_values();
void get_data_by_multiplicity(int *&Pts, int &nb_pts,
   int multiplicity, int verbose_level);
void get_data_by_multiplicity_as_lint(
      long int *&Pts, int &nb_pts,
      int multiplicity, int verbose_level);
int determine_class_by_value(int value);
int get_value_of_class(int class_idx);
```

```
   int get_largest_value();
   void get_class_by_value(int *&Pts, int &nb_pts, int value,
      int verbose_level);
   void get_class_by_value_lint(
         long int *&Pts, int &nb_pts, int value, int verbose_level);
   data_structures::set_of_sets *get_set_partition_and_types(
         int *&types,
      int &nb_types, int verbose_level);
   void save_classes_individually(std::string &fname);
};



// ##############################################################################
// tally.cpp
// ##############################################################################


//! a statistical analysis of data consisting of single integers



class tally {

public:

   int data_length;

   int f_data_ownership;
   int *data;
   int *data_sorted;
   int *sorting_perm;
      // computed using int_vec_sorting_permutation
   int *sorting_perm_inv;
      // perm_inv[i] is the index in data
      // of the element in data_sorted[i]
   int nb_types;
   int *type_first;
   int *type_len;

   int f_second;
   int *second_data_sorted;
   int *second_sorting_perm;
   int *second_sorting_perm_inv;
   int second_nb_types;
   int *second_type_first;
   int *second_type_len;
```

```
// added 09/28/2022:
data_structures::set_of_sets *Set_partition;
//data_structures::set_of_sets *Orbits_classified;

int *data_values; // [nb_types]

//int *Orbits_classified_length; // [Orbits_classified_nb_types]
//int Orbits_classified_nb_types;



tally();
~tally();
void init(int *data, int data_length,
   int f_second, int verbose_level);
void init_lint(long int *data, int data_length,
   int f_second, int verbose_level);
void sort_and_classify();
void sort_and_classify_second();
int class_of(int pt_idx);
void print(int f_backwards);
void print_no_lf(int f_backwards);
void print_tex_no_lf(int f_backwards);
void print_first(int f_backwards);
void print_second(int f_backwards);
void print_first_tex(int f_backwards);
void print_second_tex(int f_backwards);
void print_file(std::ostream &ost, int f_backwards);
void print_file_tex(std::ostream &ost, int f_backwards);
void print_file_tex_we_are_in_math_mode(
      std::ostream &ost, int f_backwards);
void print_naked_stringstream(
      std::stringstream &sstr, int f_backwards);
void print_naked(int f_backwards);
void print_naked_tex(std::ostream &ost, int f_backwards);
void print_types_naked_tex(std::ostream &ost, int f_backwards,
   int *the_vec_sorted,
   int nb_types, int *type_first, int *type_len);
void print_array_tex(std::ostream &ost, int f_backwards);
double average();
double average_of_non_zero_values();
void get_data_by_multiplicity(int *&Pts, int &nb_pts,
   int multiplicity, int verbose_level);
void get_data_by_multiplicity_as_lint(
      long int *&Pts, int &nb_pts,
      int multiplicity, int verbose_level);
```

```
   int determine_class_by_value(int value);
   int get_value_of_class(int class_idx);
   int get_largest_value();
   void get_class_by_value(int *&Pts, int &nb_pts, int value,
      int verbose_level);
   void get_class_by_value_lint(
         long int *&Pts, int &nb_pts, int value, int verbose_level);
   data_structures::set_of_sets *get_set_partition_and_types(
         int *&types,
      int &nb_types, int verbose_level);
   void save_classes_individually(std::string &fname);
};


// ##############################################################################
// tally_vector_data.cpp
// ##############################################################################


//! a statistical analysis of data consisting of vectors of ints



class tally_vector_data {

public:

   int data_set_sz;
   int data_length;

   int *data; // [data_length * data_set_sz]

   int *rep_idx;
      // [data_length],
      // rep_idx[i] is the index into Rep of data[i * data_set_sz]
   int *Reps;
      // [data_length * data_set_sz],
      // used [nb_types * data_set_sz]
   int *Frequency;
      // [data_length], used [nb_types]
   int *sorting_perm;
      // computed using int_vec_sorting_permutation
   int *sorting_perm_inv;
      // perm_inv[i] is the index in data
      // of the element in data_sorted[i]
```

```
   std::multimap<uint32_t, int> Hashing;
      // we store the pair (hash, idx)
      // where hash is the hash value of the set and idx is the
      // index in the table Sets where the set is stored.
      //
      // we use a multimap because the hash values are not unique
      // it happens that two sets have the same hash value.
      // map cannot handle that.

   int nb_types;
   int *type_first;
   //int *type_len; same as Frequency[]

   int **Reps_in_lex_order; // [nb_types]
   int *Frequency_in_lex_order; // [nb_types]


   tally_vector_data();
   ~tally_vector_data();
   void init(int *data, int data_length, int data_set_sz,
      int verbose_level);
   int hash_and_find(int *data,
         int &idx, uint32_t &h, int verbose_level);
   void print();
   void save_classes_individually(
         std::string &fname, int verbose_level);
   void get_transversal(
         int *&transversal, int *&frequency,
         int &nb_types, int verbose_level);
   void print_classes_bigger_than_one(int verbose_level);

};




// ##############################################################################
// vector_builder_description.cpp
// ##############################################################################



//! to define a vector of field elements


class vector_builder_description {
public:
```

```
int f_field;
std::string field_label;

int f_allow_negatives;

int f_dense;
std::string dense_text;

int f_compact;
std::string compact_text;

int f_repeat;
std::string repeat_text;
int repeat_length;

int f_format;
int format_k;

int f_file;
std::string file_name;

int f_load_csv_no_border;
std::string load_csv_no_border_fname;

int f_load_csv_data_column;
std::string load_csv_data_column_fname;
int load_csv_data_column_idx;

int f_sparse;
int sparse_len;
std::string sparse_pairs;

int f_concatenate;
std::vector<std::string> concatenate_list;

int f_loop;
int loop_start;
int loop_upper_bound;
int loop_increment;

vector_builder_description();
~vector_builder_description();
int read_arguments(
   int argc, std::string *argv,
   int verbose_level);
void print();
```

```
};



// ##############################################################################
// vector_builder.cpp
// ##############################################################################



//! to create a vector of field elements from class vector_builder_description


class vector_builder {
public:

   vector_builder_description *Descr;

   field_theory::finite_field *F;

   long int *v;
   int len;

   int f_has_k;
   int k;

   vector_builder();
   ~vector_builder();
   void init(vector_builder_description *Descr,
         field_theory::finite_field *F,
         int verbose_level);
   void print(std::ostream &ost);

};



// ##############################################################################
// vector_hashing.cpp
// ##############################################################################

//! hash tables


class vector_hashing {

public:
```

```
   int data_size;
   int N;
   int bit_length;
   int *vector_data;
   int *H;
   int *H_sorted;
   int *perm;
   int *perm_inv;
   int nb_types;
   int *type_first;
   int *type_len;
   int *type_value;


   vector_hashing();
   ~vector_hashing();
   void allocate(int data_size, int N, int bit_length);
   void compute_tables(int verbose_level);
   void print();
   int rank(int *data);
   void unrank(int rk, int *data);
};

}}}



#endif /* ORBITER_SRC_LIB_FOUNDATIONS_DATA_STRUCTURES_DATA_STRUCTURES_H_ */
```

3.8 Expression Parser

```
/*
 * parser.h
 *
 *  Created on: Feb 14, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_FOUNDATIONS_EXPRESSION_PARSER_EXPRESSION_PARSER_H_
#define SRC_LIB_FOUNDATIONS_EXPRESSION_PARSER_EXPRESSION_PARSER_H_



namespace orbiter {
namespace layer1_foundations {
namespace expression_parser {


// ##############################################################################
// expression_parser_domain.cpp
// ##############################################################################



//! a domain for things related to expression parsing

class expression_parser_domain {
public:
   expression_parser_domain();
   ~expression_parser_domain();
   void parse_and_evaluate(
         field_theory::finite_field *F,
         std::string &name_of_formula,
         std::string &formula_text,
         std::string &managed_variables,
         int f_evaluate,
         std::string &parameters,
         int verbose_level);
   void evaluate(
         field_theory::finite_field *Fq,
         std::string &formula_label,
         std::string &parameters,
         int verbose_level);
   int evaluate_formula(
         formula *F,
         field_theory::finite_field *Fq,
         std::string &parameters,
```

```
        int verbose_level);
   void evaluate_managed_formula(
        formula *F,
        field_theory::finite_field *Fq,
        std::string &parameters,
        int *&Values, int &nb_monomials,
        int verbose_level);

};




// ##############################################################################
// expression_parser.cpp
// ##############################################################################



//! class to parse expressions




class expression_parser {



   private:

      lexer *Lexer;

   public:

      // symbol table
      std::map<std::string, double> symbols;

   syntax_tree *Tree;

   expression_parser();
   ~expression_parser();


   // access symbols with operator []
   double & operator[] (std::string & key) { return symbols [key]; }
```

```
   syntax_tree_node *Primary(int verbose_level,
        int &f_single_literal, std::string &single_literal,
        int &f_has_seen_minus, const bool get);
   syntax_tree_node *Term(int verbose_level, const bool get);
   syntax_tree_node *AddSubtract(int verbose_level, const bool get);
   syntax_tree_node *Comparison(int verbose_level, const bool get);
   syntax_tree_node *Expression(int verbose_level, const bool get);
   syntax_tree_node *CommaList(int verbose_level, const bool get);
   void parse(syntax_tree *tree, std::string & program, int verbose_level);

};


// ##############################################################################
// formula_activity_description.cpp
// ##############################################################################


//! description of an activity involving a formula

class formula_activity_description {
public:

   int f_export;

   int f_evaluate;
   std::string evaluate_finite_field_label;
   std::string evaluate_assignment;

   int f_print_over_Fq;
   std::string print_over_Fq_field_label;

   int f_sweep;
   std::string sweep_field_label;
   std::string sweep_variables;

   int f_sweep_affine;
   std::string sweep_affine_field_label;
   std::string sweep_affine_variables;



   formula_activity_description();
   ~formula_activity_description();
   int read_arguments(
```

```
      int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// formula_activity.cpp
// ##############################################################################


//! an activity involving a formula

class formula_activity {
public:


   formula_activity_description *Descr;
   formula *f;

   formula_activity();
   ~formula_activity();
   void init(formula_activity_description *Descr,
         formula *f,
         int verbose_level);
   void perform_activity(int verbose_level);
   void do_sweep(int f_affine,
         formula *f,
         field_theory::finite_field *F, std::string &sweep_variables,
         int verbose_level);

};


// ##############################################################################
// formula.cpp
// ##############################################################################



//! front-end to expression
```

```
class formula {


public:

   std::string name_of_formula;
   std::string name_of_formula_latex;
   std::string managed_variables;
   std::string formula_text;
   syntax_tree *tree;


   int nb_managed_vars;

   int f_is_homogeneous;
   int degree;


   formula();
   ~formula();
   void print();
   void init(std::string &label, std::string &label_tex,
         std::string &managed_variables, std::string &formula_text,
         int verbose_level);
#if 0
   void init_Sajeeb(std::string &label, std::string &label_tex,
         std::string &managed_variables, std::string &formula_text,
         int verbose_level);
#endif
   int is_homogeneous(int &degree, int verbose_level);
   void get_subtrees(ring_theory::homogeneous_polynomial_domain *Poly,
         syntax_tree_node **&Subtrees, int &nb_monomials,
         int verbose_level);
   void evaluate(ring_theory::homogeneous_polynomial_domain *Poly,
         syntax_tree_node **Subtrees, std::string &evaluate_text, int *Values,
         int verbose_level);
   void print(std::ostream &ost);
   void print_easy(field_theory::finite_field *F, std::ostream &ost);

};


// ##############################################################################
// lexer.cpp
// ##############################################################################

//! lexical analysis of expressions
```

```
class lexer {
public:
     std::string program;

     const char * pWord;
     const char * pWordStart;
     // last token parsed
     TokenType type;
     std::string word;
     double value;
     syntax_tree_node_terminal *T;

     lexer();
     void print_token(std::ostream &ost, TokenType t);
     void token_as_string(std::string &s, TokenType t);
     TokenType GetToken (int verbose_level, const bool ignoreSign = false);
     void create_text_token(std::string &txt);
     void create_double_token(double dbl);
     void CheckToken (TokenType wanted);

};

// ##############################################################################
// syntax_tree_node_terminal.cpp
// ##############################################################################

//! terminal node in the syntax tree of an expression



class syntax_tree_node_terminal {
public:
   int f_int;
   int f_double;
   int f_text;
   int value_int;
   double value_double;
   std::string value_text;

   syntax_tree_node_terminal();
   void print(std::ostream &ost);
   void print_easy(std::ostream &ost);
   void print_expression(std::ostream &ost);
   void print_graphviz(std::ostream &ost);
```

```
   int evaluate(
         std::map<std::string, std::string> &symbol_table,
         field_theory::finite_field *F,
         int verbose_level);

};



// ##############################################################################
// syntax_tree_node.cpp
// ##############################################################################

#define MAX_NODES_SYNTAX_TREE 1000


//! interior node in a syntax tree


class syntax_tree_node {
public:
   syntax_tree *Tree;
   int idx;

   int f_terminal;
   syntax_tree_node_terminal *T;

   //! multiplication or addition
   enum syntax_tree_node_operation_type type;

   // ! if we are not a terminal node, we can have any number of nodes
   int nb_nodes;
   syntax_tree_node *Nodes[MAX_NODES_SYNTAX_TREE];

   int f_has_monomial; // only for multiplication nodes
   int *monomial;

   int f_has_minus;

   syntax_tree_node();
   ~syntax_tree_node();
   void null();
   void split_by_monomials(
         ring_theory::homogeneous_polynomial_domain *Poly,
         syntax_tree_node **Subtrees, int verbose_level);
   int is_homogeneous(int &degree, int verbose_level);
   void print(std::ostream &ost);
```

```
   void print_easy(std::ostream &ost);
   void print_easy_without_monomial(std::ostream &ost);
   int is_mult();
   int is_add();
   int evaluate(std::map<std::string, std::string> &symbol_table,
         field_theory::finite_field *F, int verbose_level);
   void print_expression(std::ostream &ost);
   void push_a_minus_sign();
   void print_without_recursion(std::ostream &ost);
   void export_graphviz(std::string &name, std::ostream &ost);
   void export_graphviz_recursion(std::ostream &ost);
};


// ##############################################################################
// syntax_tree.cpp
// ##############################################################################

//! the syntax tree of an expression, possibly with managed variables



class syntax_tree {
public:
   int f_has_managed_variables;
   std::vector<std::string> managed_variables;

   syntax_tree_node *Root;

   syntax_tree();
   void print(std::ostream &ost);
   void print_easy(std::ostream &ost);
   void print_monomial(std::ostream &ost, int *monomial);
   int identify_single_literal(std::string &single_literal);
   int is_homogeneous(int &degree, int verbose_level);
   void split_by_monomials(
         ring_theory::homogeneous_polynomial_domain *Poly,
         syntax_tree_node **&Subtrees, int verbose_level);

};
```

```
}}}


#endif /* SRC_LIB_FOUNDATIONS_EXPRESSION_PARSER_EXPRESSION_PARSER_H_ */
```

3.9 Finite Fields

```
/*
 * finite_fields.h
 *
 *  Created on: Mar 2, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_FOUNDATIONS_FINITE_FIELDS_FINITE_FIELDS_H_
#define SRC_LIB_FOUNDATIONS_FINITE_FIELDS_FINITE_FIELDS_H_

namespace orbiter {
namespace layer1_foundations {
namespace field_theory {



// ##############################################################################
// finite_field_activity_description.cpp
// ##############################################################################


//! description of a finite field activity

class finite_field_activity_description {
public:

   int f_cheat_sheet_GF;

   int f_export_tables;


   int f_polynomial_division;
   std::string polynomial_division_A;
   std::string polynomial_division_B;

   int f_extended_gcd_for_polynomials;

   int f_polynomial_mult_mod;
   std::string polynomial_mult_mod_A;
   std::string polynomial_mult_mod_B;
   std::string polynomial_mult_mod_M;

   int f_polynomial_power_mod;
   std::string polynomial_power_mod_A;
   std::string polynomial_power_mod_n;
   std::string polynomial_power_mod_M;
```

```
int f_Berlekamp_matrix;
std::string Berlekamp_matrix_label;

int f_polynomial_find_roots;
std::string polynomial_find_roots_label;


int f_product_of;
std::string product_of_elements;

int f_sum_of;
std::string sum_of_elements;

int f_negate;
std::string negate_elements;

int f_inverse;
std::string inverse_elements;

int f_power_map;
int power_map_k;
std::string power_map_elements;

int f_parse_and_evaluate;
std::string parse_name_of_formula;
std::string parse_managed_variables;
std::string parse_text;
std::string parse_parameters;

int f_evaluate;
std::string evaluate_formula_label;
std::string evaluate_parameters;



// Section 3.3:
// Extension fields:

int f_trace;

int f_norm;

int f_normal_basis;
int normal_basis_d;

int f_nth_roots;
```

```
int nth_roots_n;

int f_field_reduction;
std::string field_reduction_label;
int field_reduction_q;
int field_reduction_m;
int field_reduction_n;
std::string field_reduction_text;



// Section 3.4:
// Linear algebra:

int f_nullspace;
std::string nullspace_input_matrix;

int f_RREF;
std::string RREF_input_matrix;

// the following two options affect nullspace and RREF:
int f_normalize_from_the_right;
int f_normalize_from_the_left;


int f_RREF_random_matrix;
int RREF_random_matrix_m;
int RREF_random_matrix_n;

int f_Walsh_matrix;
int Walsh_matrix_n;

int f_Vandermonde_matrix;






int f_Walsh_Hadamard_transform;
std::string Walsh_Hadamard_transform_fname_csv_in;
int Walsh_Hadamard_transform_n;

int f_algebraic_normal_form_of_boolean_function;
std::string algebraic_normal_form_of_boolean_function_fname_csv_in;
int algebraic_normal_form_of_boolean_function_n;
```

```
int f_algebraic_normal_form;
int algebraic_normal_form_n;
std::string algebraic_normal_form_input;

int f_apply_trace_function;
std::string apply_trace_function_fname_csv_in;

int f_apply_power_function;
std::string apply_power_function_fname_csv_in;
long int apply_power_function_d;

int f_identity_function;
std::string identity_function_fname_csv_out;

int f_search_APN_function;


int f_make_table_of_irreducible_polynomials;
int make_table_of_irreducible_polynomials_degree;

int f_get_primitive_polynomial;
int get_primitive_polynomial_degree;

int f_get_primitive_polynomial_in_range;
int get_primitive_polynomial_in_range_min;
int get_primitive_polynomial_in_range_max;


// cryptography:

int f_EC_Koblitz_encoding;
std::string EC_message;
int EC_s;
// EC_b, EC_c, EC_s, EC_pt_text, EC_message

int f_EC_points;
std::string EC_label;

int f_EC_add;
std::string EC_pt1_text;
std::string EC_pt2_text;

int f_EC_cyclic_subgroup;
int EC_b;
int EC_c;
std::string EC_pt_text;
```

```
int f_EC_multiple_of;
int EC_multiple_of_n;


int f_EC_discrete_log;
std::string EC_discrete_log_pt_text;
// EC_b, EC_c, EC_pt_text, EC_discrete_log_pt_text


int f_EC_baby_step_giant_step;
std::string EC_bsgs_G;
int EC_bsgs_N;
std::string EC_bsgs_cipher_text;

int f_EC_baby_step_giant_step_decode;
std::string EC_bsgs_A;
std::string EC_bsgs_keys;

int f_NTRU_encrypt;
int NTRU_encrypt_N;
int NTRU_encrypt_p;
std::string NTRU_encrypt_H;
std::string NTRU_encrypt_R;
std::string NTRU_encrypt_Msg;

int f_polynomial_center_lift;
std::string polynomial_center_lift_A;

int f_polynomial_reduce_mod_p;
std::string polynomial_reduce_mod_p_A;


int f_cheat_sheet_hermitian;
int cheat_sheet_hermitian_projective_dimension;

int f_cheat_sheet_desarguesian_spread;
int cheat_sheet_desarguesian_spread_m;

int f_sift_polynomials;
long int sift_polynomials_r0;
long int sift_polynomials_r1;

int f_mult_polynomials;
long int mult_polynomials_r0;
long int mult_polynomials_r1;

int f_polynomial_division_ranked;
```

```
   long int polynomial_division_r0;
   long int polynomial_division_r1;



   int f_transversal;
   std::string transversal_line_1_basis;
   std::string transversal_line_2_basis;
   std::string transversal_point;

   int f_intersection_of_two_lines;
   std::string line_1_basis;
   std::string line_2_basis;

   int f_inverse_isomorphism_klein_quadric;
   std::string inverse_isomorphism_klein_quadric_matrix_A6;

   // ranking and unranking points in PG:

   int f_rank_point_in_PG;
   std::string rank_point_in_PG_label;

   int f_unrank_point_in_PG;
   int unrank_point_in_PG_n;
   std::string unrank_point_in_PG_text;





   finite_field_activity_description();
   ~finite_field_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// finite_field_activity.cpp
// ##############################################################################
```

```
//! perform a finite field activity

class finite_field_activity {
public:
   finite_field_activity_description *Descr;
   finite_field *F;
   finite_field *F_secondary;

   finite_field_activity();
   ~finite_field_activity();
   void init(finite_field_activity_description *Descr,
         finite_field *F,
         int verbose_level);
   void perform_activity(int verbose_level);

};


// ##############################################################################
// finite_field_implementation_by_tables.cpp
// ##############################################################################

//! implementation of a finite Galois field Fq using tables

class finite_field_implementation_by_tables {


private:

   finite_field *F;

   int *add_table; // [q * q]
   int *mult_table; // [q * q]
      // add_table and mult_table are needed in mindist

   int *negate_table; // [q]
   int *inv_table; // [q]
   int *frobenius_table; // [q], x \mapsto x^p
   int *absolute_trace_table; // [q]
   int *log_alpha_table; // [q]
   // log_alpha_table[i] = the integer k s.t. alpha^k = i (if i > 0)
   // log_alpha_table[0] = -1
   int *alpha_power_table; // [q]

   int *v1; // [e]
   int *v2; // [e]
   int *v3; // [e]
```

```
   int f_has_quadratic_subfield; // TRUE if e is even.
   int *f_belongs_to_quadratic_subfield; // [q]

   int *reordered_list_of_elements; // [q]
   int *reordered_list_of_elements_inv; // [q]

public:

   finite_field_implementation_by_tables();
   ~finite_field_implementation_by_tables();
   void init(finite_field *F, int verbose_level);
   int *private_add_table();
   int *private_mult_table();
   int has_quadratic_subfield();
   int belongs_to_quadratic_subfield(int a);
   void create_alpha_table(int verbose_level);
   void create_alpha_table_prime_field(int verbose_level);
   void create_alpha_table_extension_field(int verbose_level);
   void init_binary_operations(int verbose_level);
   void create_tables_prime_field(int verbose_level);
   void create_tables_extension_field(int verbose_level);
   void print_add_mult_tables(std::ostream &ost);
   void print_add_mult_tables_in_C(std::string &fname_base);
   void init_quadratic_subfield(int verbose_level);
   void init_frobenius_table(int verbose_level);
   void init_absolute_trace_table(int verbose_level);
   void print_tables_extension_field(std::string &poly);
   int add(int i, int j);
   int add_without_table(int i, int j);
   int mult_verbose(int i, int j, int verbose_level);
   int mult_using_discrete_log(int i, int j, int verbose_level);
   int negate(int i);
   int negate_without_table(int i);
   int inverse(int i);
   int inverse_without_table(int i);
   int frobenius_image(int a);
   // computes a^p
   int frobenius_power(int a, int frob_power);
   // computes a^{p^frob_power}
   int alpha_power(int i);
   int log_alpha(int i);
   void addition_table_reordered_save_csv(
         std::string &fname, int verbose_level);
   void multiplication_table_reordered_save_csv(
         std::string &fname, int verbose_level);
```

```
};


// ##############################################################################
// finite_field_implementation_wo_tables.cpp
// ##############################################################################

//! implementation of a finite Galois field Fq without any tables

class finite_field_implementation_wo_tables {


private:

   finite_field *F;

   int *v1; // [e]
   int *v2; // [e]
   int *v3; // [e]

   finite_field *GFp;
      // only allocated if e > 1
      // (otherwise we would be an infinite recursion)

   ring_theory::unipoly_domain *FX;

   ring_theory::unipoly_object m;

   int factor_polynomial_degree;
   int *factor_polynomial_coefficients_negated;


   ring_theory::unipoly_domain *Fq;

   ring_theory::unipoly_object Alpha;


public:
   finite_field_implementation_wo_tables();
   ~finite_field_implementation_wo_tables();
   void init(finite_field *F, int verbose_level);
   void init_extension_field(int verbose_level);
   int mult(int i, int j, int verbose_level);
   int inverse(int i, int verbose_level);
   int negate(int i, int verbose_level);
   int add(int i, int j, int verbose_level);
```

```
};



// ##############################################################################
// finite_field_description.cpp
// ##############################################################################


//! description of a finite field

class finite_field_description {
public:

   int f_q;
   std::string q_text;

   int f_override_polynomial;
   std::string override_polynomial;

   int f_without_tables;

   int f_compute_related_fields;

   int f_symbol;
   std::string symbol_label;

   finite_field_description();
   ~finite_field_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// finite_field.cpp
// ##############################################################################

//! finite field Fq

class finite_field {

private:
   finite_field_implementation_by_tables *T;
```

```
   finite_field_implementation_wo_tables *Iwo;


   std::string symbol_for_print;


   int nb_times_mult;
   int nb_times_add;

public:

   finite_field_description *Descr;

   int f_has_table;
      // if TRUE, T is available, otherwise Iwo is available.

   std::string label;
   std::string label_tex;
   std::string override_poly;
   std::string my_poly; // numerical value as text
   std::string my_poly_tex; // pretty printed polynomial
   ring_theory::longinteger_object *my_poly_longinteger;
   long int my_poly_lint;
   int *my_poly_vec;
   //char *polynomial;
      // the actual polynomial we consider
      // as integer (in text form)
   int f_is_prime_field;

   std::string q_text;
   ring_theory::longinteger_object *q_longinteger;
   long int q_long;

   int q;
   int p, e; // q = p^e

   int alpha; // primitive element
   int log10_of_q; // needed for printing purposes
   int f_print_as_exponentials;
   long int nb_calls_to_mult_matrix_matrix;
   long int nb_calls_to_PG_element_rank_modified;
   long int nb_calls_to_PG_element_unrank_modified;

   linear_algebra::linear_algebra *Linear_algebra;

   int f_related_fields_have_been_computed;
```

```
related_fields *Related_fields;


finite_field();
~finite_field();
void print_call_stats(std::ostream &ost);
void init(
      finite_field_description *Descr,
      int verbose_level);
void finite_field_init(
      std::string &q_text,
      int f_without_tables,
      int f_compute_related_fields,
      int verbose_level);
void check_size(int verbose_level);
void finite_field_init_small_order(int q,
      int f_without_tables,
      int f_compute_related_fields,
      int verbose_level);
void setup_related_fields(int f_compute_related_fields,
      int verbose_level);
void init_override_polynomial(
      std::string &q_text,
      std::string &poly,
      int f_without_tables,
      int f_compute_related_fields,
      int verbose_level);
void init_override_polynomial_small_order(
      int q,
      std::string &poly,
      int f_without_tables,
      int f_compute_related_fields,
      int verbose_level);

void init_implementation(
      int f_without_tables, int verbose_level);
void set_default_symbol_for_print();
void init_symbol_for_print(std::string &symbol);
int has_quadratic_subfield();
int belongs_to_quadratic_subfield(int a);
long int compute_subfield_polynomial(
      int order_subfield,
      int verbose_level);
void compute_subfields(int verbose_level);
int find_primitive_element(int verbose_level);
int compute_order_of_element(
      int elt, int verbose_level);
```

```
int *private_add_table();
int *private_mult_table();
int zero();
int one();
int minus_one();
int is_zero(int i);
int is_one(int i);
int mult(int i, int j);
int mult_verbose(int i, int j, int verbose_level);
int a_over_b(int a, int b);
int mult3(int a1, int a2, int a3);
int product3(int a1, int a2, int a3);
int mult4(int a1, int a2, int a3, int a4);
int mult5(int a1, int a2, int a3, int a4, int a5);
int mult6(int a1, int a2, int a3, int a4, int a5, int a6);
int product4(int a1, int a2, int a3, int a4);
int product5(int a1, int a2, int a3, int a4, int a5);
int product_n(int *a, int n);
int square(int a);
int twice(int a);
int four_times(int a);
int Z_embedding(int k);
int add(int i, int j);
int add3(int i1, int i2, int i3);
int add4(int i1, int i2, int i3, int i4);
int add5(int i1, int i2, int i3, int i4, int i5);
int add6(int i1, int i2, int i3, int i4, int i5, int i6);
int add7(int i1, int i2, int i3, int i4, int i5, int i6,
   int i7);
int add8(int i1, int i2, int i3, int i4, int i5, int i6,
   int i7, int i8);
int negate(int i);
int inverse(int i);
int power(int a, int n);
int power_verbose(int a, int n, int verbose_level);
   // computes a^n
void frobenius_power_vec(
      int *v, int len, int frob_power);
void frobenius_power_vec_to_vec(int *v_in, int *v_out,
      int len, int frob_power);
int frobenius_power(int a, int frob_power);
   // computes a^{p^frob_power}
int absolute_trace(int i);
int absolute_norm(int i);
int alpha_power(int i);
int log_alpha(int i);
int multiplicative_order(int a);
```

```
void all_square_roots(int a, int &nb_roots, int *roots2);
int is_square(int i);
int square_root(int i);
int primitive_root();
int N2(int a);
int N3(int a);
int T2(int a);
int T3(int a);
int bar(int a);
void abc2xy(
      int a, int b, int c, int &x, int &y,
   int verbose_level);
   // given a, b, c, determine x and y such that
   // c = a * x^2 + b * y^2
   // such elements x and y exist for any choice of a, b, c.
int nb_times_mult_called();
int nb_times_add_called();
void compute_nth_roots(
      int *&Nth_roots, int n, int verbose_level);
int primitive_element();



// ##########################################################################
// finite_field_projective.cpp
// ##########################################################################

void PG_element_apply_frobenius(
      int n, int *v, int f);
int test_if_vectors_are_projectively_equal(
      int *v1, int *v2, int len);
void PG_element_normalize(
      int *v, int stride, int len);
// last non-zero element made one
void PG_element_normalize_from_front(
      int *v, int stride, int len);
// first non zero element made one
void PG_element_normalize_from_a_given_position(
      int *v, int stride, int len, int idx);


void PG_elements_embed(
      long int *set_in, long int *set_out, int sz,
      int old_length, int new_length, int *v);
long int PG_element_embed(
      long int rk, int old_length, int new_length, int *v);
void PG_element_unrank_fining(
```

```
      int *v, int len, int a);
int PG_element_rank_fining(
      int *v, int len);
void PG_element_unrank_gary_cook(
      int *v, int len, int a);
void PG_element_rank_modified(
      int *v, int stride, int len, int &a);
void PG_element_unrank_modified(
      int *v, int stride, int len, int a);
void PG_element_rank_modified_lint(
      int *v, int stride, int len, long int &a);
void PG_elements_unrank_lint(
      int *M, int k, int n, long int *rank_vec);
void PG_elements_rank_lint(
      int *M, int k, int n, long int *rank_vec);
void PG_element_unrank_modified_lint(
      int *v, int stride, int len, long int a);
void PG_element_rank_modified_not_in_subspace(
      int *v, int stride, int len, int m, long int &a);
void PG_element_unrank_modified_not_in_subspace(
      int *v, int stride, int len, int m, long int a);

void projective_point_unrank(int n, int *v, int rk);
long int projective_point_rank(int n, int *v);
void all_PG_elements_in_subspace(
      int *genma, int k, int n,
      long int *&point_list, int &nb_points,
      int verbose_level);
void all_PG_elements_in_subspace_array_is_given(
      int *genma, int k, int n,
      long int *point_list, int &nb_points,
      int verbose_level);
void display_all_PG_elements(int n);
void display_all_PG_elements_not_in_subspace(int n, int m);
void display_all_AG_elements(int n);



// ##########################################################################
// finite_field_io.cpp
// ##########################################################################

void report(
      std::ostream &ost, int verbose_level);
void print_minimum_polynomial_to_str(
      int p,
      std::string &polynomial, std::stringstream &s);
```

```
void print();
void print_detailed(int f_add_mult_table);
void print_tables();
void display_T2(std::ostream &ost);
void display_T3(std::ostream &ost);
void display_N2(std::ostream &ost);
void display_N3(std::ostream &ost);
void print_integer_matrix_zech(
      std::ostream &ost,
   int *p, int m, int n);
void print_indicator_square_nonsquare(int a);
void print_element(
      std::ostream &ost, int a);
void print_element_str(
      std::stringstream &ost, int a);
void print_element_with_symbol(
      std::ostream &ost,
   int a, int f_exponential, int width, std::string &symbol);
void print_element_with_symbol_str(
      std::stringstream &ost,
      int a, int f_exponential, int width, std::string &symbol);
void int_vec_print_field_elements(
      std::ostream &ost, int *v, int len);
void int_vec_print_elements_exponential(
      std::ostream &ost,
   int *v, int len, std::string &symbol_for_print);
void make_fname_addition_table_csv(
      std::string &fname);
void make_fname_multiplication_table_csv(
      std::string &fname);
void make_fname_addition_table_reordered_csv(
      std::string &fname);
void make_fname_multiplication_table_reordered_csv(
      std::string &fname);
void addition_table_save_csv(
      int verbose_level);
void multiplication_table_save_csv(
      int verbose_level);
void addition_table_reordered_save_csv(
      int verbose_level);
void multiplication_table_reordered_save_csv(
      int verbose_level);
void latex_addition_table(
      std::ostream &f,
   int f_elements_exponential, std::string &symbol_for_print);
void latex_multiplication_table(
      std::ostream &f,
```

```
   int f_elements_exponential, std::string &symbol_for_print);
void latex_matrix(
      std::ostream &f, int f_elements_exponential,
      std::string &symbol_for_print, int *M, int m, int n);
void power_table(
      int t, int *power_table, int len);
void cheat_sheet(
      std::ostream &f, int verbose_level);
void cheat_sheet_subfields(
      std::ostream &f, int verbose_level);
void report_subfields(
      std::ostream &f, int verbose_level);
void report_subfields_detailed(
      std::ostream &ost, int verbose_level);
void cheat_sheet_addition_table(
      std::ostream &f, int verbose_level);
void cheat_sheet_multiplication_table(
      std::ostream &f, int verbose_level);
void cheat_sheet_power_table(
      std::ostream &f,
      int f_with_polynomials, int verbose_level);
void cheat_sheet_power_table_top(
      std::ostream &ost,
      int f_with_polynomials, int verbose_level);
void cheat_sheet_power_table_bottom(
      std::ostream &ost,
      int f_with_polynomials, int verbose_level);
void cheat_sheet_table_of_elements(
      std::ostream &ost, int verbose_level);
void print_element_as_polynomial(
      std::ostream &ost, int *v, int verbose_level);
void cheat_sheet_main_table(
      std::ostream &f, int verbose_level);
void cheat_sheet_main_table_top(
      std::ostream &f, int nb_cols);
void cheat_sheet_main_table_bottom(
      std::ostream &f);
void display_table_of_projective_points(
      std::ostream &ost, long int *Pts, int nb_pts, int len);
void display_table_of_projective_points2(
   std::ostream &ost, long int *Pts, int nb_pts, int len);
void display_table_of_projective_points_easy(
   std::ostream &ost, long int *Pts, int nb_pts, int len);
void print_matrix_latex(
      std::ostream &ost, int *A, int m, int n);
void print_matrix_numerical_latex(
      std::ostream &ost, int *A, int m, int n);
```

```
   void read_from_string_coefficient_vector(
         std::string &str,
         int *&coeff, int &len,
         int verbose_level);



};


// ##############################################################################
// minimum_polynomial.cpp:
// ##############################################################################

//! to compute the minimum polynomial of a field element in an extension field

class minimum_polynomial {
public:

   finite_field *F;

   int order_subfield;
   int subgroup_index;
   int e1;
   int q1;

   int *M; // [F->e * (e1 + 1)]
   int *K;
   int *base_cols;
   int kernel_m, kernel_n;

   long int min_poly_rank;

   std::string min_poly_rank_as_string;

   minimum_polynomial();
   ~minimum_polynomial();
   void compute_subfield_polynomial(
         finite_field *F,
         int order_subfield, int verbose_level);
   void report_table(std::ostream &ost);

};
```

```
// ##############################################################################
// norm_tables.cpp:
// ##############################################################################

//! tables for the norm map in a finite field

class norm_tables {
public:
   int *norm_table;
   int *norm_table_sorted;
   int *sorting_perm, *sorting_perm_inv;
   int nb_types;
   int *type_first, *type_len;
   int *the_type;

   norm_tables();
   ~norm_tables();
   void init(
         orthogonal_geometry::unusual_model &U,
         int verbose_level);
   int choose_an_element_of_given_norm(
         int norm, int verbose_level);

};


// ##############################################################################
// nth_roots.cpp:
// ##############################################################################

//! the nth roots over Fq using an extension field

class nth_roots {
public:

   int n;
   finite_field *F; // F_q where q = p^e
   ring_theory::unipoly_object *Beta;
   ring_theory::unipoly_object *Fq_Elements;

   ring_theory::unipoly_object Min_poly;
      // Min_poly = irreducible polynomial
      // over F->p of degree field_degree = e * m

   finite_field *Fp; // the prime field F_p

   ring_theory::unipoly_domain *FpX;
```

```
      // polynomial ring over Fp (the small field)

   ring_theory::unipoly_domain *FQ;
      // polynomial ring F_p modulo Min_poly

   ring_theory::unipoly_domain *FX;
      // polynomial ring over F (the big field)

   int m, r, field_degree;
      // m is the order of q modulo n
      // field_degree = e * m

   ring_theory::longinteger_object *Qm, *Qm1, *Index, *Subfield_Index;
      // Qm = q^m = p^(e*m)
      // Qm1 = q^m - 1
      // Index = Qm1 / n
      // Subfield_Index = Qm1 / (q - 1)

   number_theory::cyclotomic_sets *Cyc;
   ring_theory::unipoly_object **min_poly_beta_FQ;
   // polynomials whose coefficients are again polynomials,
   // representing field elements in FQ
   //ring_theory::unipoly_object **generator;

   ring_theory::unipoly_object *min_poly_beta_Fq;
   // polynomials whose coefficients are integers in Fq
   //ring_theory::unipoly_object *generator_Fq;

   int subfield_degree;
   int *subfield_basis; // [subfield_degree * field_degree]





   nth_roots();
   ~nth_roots();
   void init(
         finite_field *F, int n, int verbose_level);
   void compute_subfield(int subfield_degree,
         int *&field_basis, int verbose_level);
   void report(std::ostream &ost, int verbose_level);

};
```

```
// ##############################################################################
// related_fields.cpp
// ##############################################################################

//! fields that are related to a given field Fq

class related_fields {

private:

public:

   finite_field *F;


   int nb_subfields;
   int *Subfield_order; // [nb_subfields]
   int *Subfield_exponent; // [nb_subfields]
   int *Subfield_index; // [nb_subfields]
   minimum_polynomial *Subfield_minimum_polynomial;
      // [nb_subfields]

   finite_field *Subfield; // [nb_subfields]

   subfield_structure *SubS; // [nb_subfields]

   related_fields();
   ~related_fields();
   void init(
         finite_field *F, int verbose_level);
   void print(std::ostream &ost);
   int position_of_subfield(int order);

};



// ##############################################################################
// square_nonsquare.cpp:
// ##############################################################################

//! keeping track of squares and nonsquares

class square_nonsquare {
public:

   finite_field *F;
```

```
   int *minus_squares; // [(q-1)/2]
   int *minus_squares_without; // [(q-1)/2 - 1]
   int *minus_nonsquares; // [(q-1)/2]
   int *f_is_minus_square; // [q]
   int *index_minus_square; // [q]
   int *index_minus_square_without; // [q]
   int *index_minus_nonsquare; // [q]

   square_nonsquare();
   ~square_nonsquare();
   void init(
         field_theory::finite_field *F, int verbose_level);
   int is_minus_square(int i);
   void print_minus_square_tables();

};



// ##############################################################################
// subfield_structure.cpp:
// ##############################################################################

//! a finite field as a vector space over a subfield

class subfield_structure {
public:

   finite_field *FQ;
   finite_field *Fq;
   int Q;
   int q;
   int s; // subfield index: q^s = Q
   int index_in_multiplicative_group;
      // = (Q - 1) / (q - 1);

   int *Basis;
      // [s], entries are elements in FQ
      // Basis[i] = FQ->power(omega, i);
      // where omega = FQ->power(alpha, s);
      // and alpha = FQ->p the primitive element in FQ.

   int *embedding;
      // [Q], entries are elements in FQ,
      // indexed by elements in AG(s,q)
   int *embedding_inv;
```

```
   // [Q], entries are ranks of elements in AG(s,q),
   // indexed by elements in FQ
   // the inverse of embedding

int *components;
   // [Q * s], entries are elements in Fq
   // the vectors corresponding to the AG(s,q)
   // ranks in embedding_inv[]

int *FQ_embedding;
   // [q] entries are elements in FQ corresponding to
   // the elements in Fq
int *Fq_element;
   // [Q], entries are the elements in Fq
   // corresponding to a given FQ element
   // or -1 if the FQ element does not belong to Fq.
int *v; // [s]

// if two dimensional, i.e. Q = q^2
int f_has_2D;
int *components_2D;
int *embedding_2D;
int *pair_embedding_2D;


subfield_structure();
~subfield_structure();
void init(
      finite_field *FQ,
      finite_field *Fq,
      int verbose_level);
void init_with_given_basis(
      finite_field *FQ,
      finite_field *Fq,
   int *given_basis, int verbose_level);
int embed(int b, int verbose_level);
int retract(int b, int verbose_level);
void embed_int_vec(
         int *v_in, int *v_out, int len,
         int verbose_level);
void retract_int_vec(
         int *v_in, int *v_out, int len,
         int verbose_level);
void print_embedding();
void report(std::ostream &ost);
void report_embedding(std::ostream &ost);
void report_embedding_reverse(std::ostream &ost);
```

```
   int evaluate_over_FQ(int *v);
   int evaluate_over_Fq(int *v);
   void lift_matrix(
         int *MQ, int m, int *Mq, int verbose_level);
      // input is MQ[m * m] over the field FQ.
      // output is Mq[n * n] over the field Fq,
   void retract_matrix(int *Mq, int n, int *MQ, int m,
      int verbose_level);
      // input is Mq[n * n] over the field Fq,
      // output is MQ[m * m] over the field FQ.
   void Adelaide_hyperoval(
         long int *&Pts, int &nb_pts, int verbose_level);
   void create_adelaide_hyperoval(
         std::string &fname, int &nb_pts, long int *&Pts,
      int verbose_level);
   void field_reduction(
         int *input, int sz, int *output,
         int verbose_level);
   // input[sz], output[s * (sz * n)],
   void embedding_2dimensional(int verbose_level);
      // we think of FQ as two dimensional vector space
      // over Fq with basis (1,alpha)
      // for i,j \in Fq, with x = i + j * alpha \in FQ, we have
      // pair_embedding_2D[i * q + j] = x;
      // also,
      // components_2D[x * 2 + 0] = i;
      // components_2D[x * 2 + 1] = j;
      // also, for i \in Fq, embedding[i] is the element
      // in FQ that corresponds to i

      // components_2D[Q * 2]
      // embedding_2D[q]
      // pair_embedding_2D[q * q]
   void print_embedding_2D();
   void print_embedding_2D_table_tex();

};



}}}



#endif /* SRC_LIB_FOUNDATIONS_FINITE_FIELDS_FINITE_FIELDS_H_ */
```

3.10 Geometry

```
// geometry.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005



#ifndef ORBITER_SRC_LIB_FOUNDATIONS_GEOMETRY_GEOMETRY_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_GEOMETRY_GEOMETRY_H_


namespace orbiter {
namespace layer1_foundations {
namespace geometry {


// ##############################################################################
// andre_construction.cpp
// ##############################################################################

//! Andre / Bruck / Bose construction of a translation plane from a spread


class andre_construction {
public:
   int order; // = q^k
   int spread_size; // order + 1
   int n; // = 2 * k
   int k;
   int q;
   int N; // order^2 + order + 1


   grassmann *Grass;
   field_theory::finite_field *F;

   long int *spread_elements_numeric; // [spread_size]
   long int *spread_elements_numeric_sorted; // [spread_size]

   long int *spread_elements_perm;
   long int *spread_elements_perm_inv;
```

```
   int *spread_elements_genma; // [spread_size * k * n]
   int *pivot; //[spread_size * k]
   int *non_pivot; //[spread_size * (n - k)]


   andre_construction();
   ~andre_construction();
   void init(
         field_theory::finite_field *F,
         int k, long int *spread_elements_numeric,
      int verbose_level);
   void points_on_line(
         andre_construction_line_element *Line,
      int *pts_on_line, int verbose_level);
   void report(
         std::ostream &ost, int verbose_level);

};




// ##############################################################################
// andre_construction_point_element.cpp
// ##############################################################################


//! a point in the projective plane created using the Andre construction


class andre_construction_point_element {
public:
   andre_construction *Andre;
   int k, n, q, spread_size;
   field_theory::finite_field *F;
   int point_rank;
   int f_is_at_infinity;
   int at_infinity_idx;
   int affine_numeric;
   int *coordinates; // [n]

   andre_construction_point_element();
   ~andre_construction_point_element();
   void init(andre_construction *Andre, int verbose_level);
   void unrank(int point_rank, int verbose_level);
   int rank(int verbose_level);
};
```

```
// ##############################################################################
// andre_construction_line_element.cpp
// ##############################################################################


//! a line in the projective plane created using the Andre construction


class andre_construction_line_element {
public:
   andre_construction *Andre;
   int k, n, q, spread_size;
   field_theory::finite_field *F;
   int line_rank;
   int f_is_at_infinity;
   int affine_numeric;
   int parallel_class_idx;
   int coset_idx;
   int *pivots; // [k]
   int *non_pivots; // [n - k]
   int *coset; // [n - k]
   int *coordinates; // [(k + 1) * n], last row is special vector

   andre_construction_line_element();
   ~andre_construction_line_element();
   void init(
         andre_construction *Andre, int verbose_level);
   void unrank(int line_rank, int verbose_level);
   int rank(int verbose_level);
   int make_affine_point(int idx, int verbose_level);
      // 0 \le idx \le order
};


// ##############################################################################
// arc_basic.cpp
// ##############################################################################

//! arcs, ovals, hyperovals etc. in projective planes


class arc_basic {
public:

   field_theory::finite_field *F;
```

```
   arc_basic();
   ~arc_basic();
   void init(field_theory::finite_field *F, int verbose_level);

   void Segre_hyperoval(
         long int *&Pts, int &nb_pts, int verbose_level);
   void GlynnI_hyperoval(
         long int *&Pts, int &nb_pts, int verbose_level);
   void GlynnII_hyperoval(
         long int *&Pts, int &nb_pts, int verbose_level);
   void Subiaco_oval(
         long int *&Pts, int &nb_pts, int f_short, int verbose_level);
      // following Payne, Penttila, Pinneri:
      // Isomorphisms Between Subiaco q-Clan Geometries,
      // Bull. Belg. Math. Soc. 2 (1995) 197-222.
      // formula (53)
   void Subiaco_hyperoval(
         long int *&Pts, int &nb_pts, int verbose_level);
   int OKeefe_Penttila_32(int t);
   int Subiaco64_1(int t);
   int Subiaco64_2(int t);
   int Adelaide64(int t);
   void LunelliSce(int *pts18, int verbose_level);
   int LunelliSce_evaluate_cubic1(int *v);
      // computes X^3 + Y^3 + Z^3 + \eta^3 XYZ
   int LunelliSce_evaluate_cubic2(int *v);
      // computes X^3 + Y^3 + Z^3 + \eta^{12} XYZ

};



// ##############################################################################
// arc_in_projective_space.cpp
// ##############################################################################

//! arcs, ovals, hyperovals etc. in projective planes


class arc_in_projective_space {
public:

   projective_space *P;

   arc_in_projective_space();
   ~arc_in_projective_space();
```

```
void init(projective_space *P, int verbose_level);

void PG_2_8_create_conic_plus_nucleus_arc_1(long int *the_arc, int &size,
   int verbose_level);
void PG_2_8_create_conic_plus_nucleus_arc_2(long int *the_arc, int &size,
   int verbose_level);
void create_Maruta_Hamada_arc(long int *the_arc, int &size,
   int verbose_level);
void create_Maruta_Hamada_arc2(long int *the_arc, int &size,
   int verbose_level);
void create_pasch_arc(long int *the_arc, int &size, int verbose_level);
void create_Cheon_arc(long int *the_arc, int &size, int verbose_level);
void create_regular_hyperoval(long int *the_arc, int &size,
   int verbose_level);
void create_translation_hyperoval(long int *the_arc, int &size,
   int exponent, int verbose_level);
void create_Segre_hyperoval(long int *the_arc, int &size,
   int verbose_level);
void create_Payne_hyperoval(long int *the_arc, int &size,
   int verbose_level);
void create_Cherowitzo_hyperoval(long int *the_arc, int &size,
   int verbose_level);
void create_OKeefe_Penttila_hyperoval_32(long int *the_arc, int &size,
   int verbose_level);

void arc_lifting_diophant(
   long int *arc, int arc_sz,
   int target_sz, int target_d,
   solvers::diophant *&D,
   int verbose_level);
void arc_with_given_set_of_s_lines_diophant(
      long int *s_lines, int nb_s_lines,
      int target_sz, int arc_d, int arc_d_low, int arc_s,
      int f_dualize,
      solvers::diophant *&D,
      int verbose_level);
void arc_with_two_given_line_sets_diophant(
      long int *s_lines, int nb_s_lines, int arc_s,
      long int *t_lines, int nb_t_lines, int arc_t,
      int target_sz, int arc_d, int arc_d_low,
      int f_dualize,
      solvers::diophant *&D,
      int verbose_level);
void arc_with_three_given_line_sets_diophant(
      long int *s_lines, int nb_s_lines, int arc_s,
      long int *t_lines, int nb_t_lines, int arc_t,
      long int *u_lines, int nb_u_lines, int arc_u,
```

```
      int target_sz, int arc_d, int arc_d_low,
      int f_dualize,
      solvers::diophant *&D,
      int verbose_level);
void maximal_arc_by_diophant(
      int arc_sz, int arc_d,
      std::string &secant_lines_text,
      std::string &external_lines_as_subset_of_secants_text,
      solvers::diophant *&D,
      int verbose_level);
void arc_lifting1(
      int arc_size,
      int arc_d,
      int arc_d_low,
      int arc_s,
      std::string arc_input_set,
      std::string arc_label,
      int verbose_level);
void arc_lifting2(
      int arc_size,
      int arc_d,
      int arc_d_low,
      int arc_s,
      std::string arc_input_set,
      std::string arc_label,
      int arc_t,
      std::string t_lines_string,
      int verbose_level);
void arc_lifting3(
      int arc_size,
      int arc_d,
      int arc_d_low,
      int arc_s,
      std::string arc_input_set,
      std::string arc_label,
      int arc_t,
      std::string t_lines_string,
      int arc_u,
      std::string u_lines_string,
      int verbose_level);
void create_hyperoval(
   int f_translation, int translation_exponent,
   int f_Segre, int f_Payne, int f_Cherowitzo, int f_OKeefe_Penttila,
   std::string &label_txt,
   std::string &label_tex,
   int &nb_pts, long int *&Pts,
   int verbose_level);
```

```
   void create_subiaco_oval(
      int f_short,
      std::string &label_txt,
      std::string &label_tex,
      int &nb_pts, long int *&Pts,
      int verbose_level);
   void create_subiaco_hyperoval(
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
         int verbose_level);
   void create_Maruta_Hamada_arc(
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
      int verbose_level);
   int arc_test(long int *input_pts, int nb_pts,
      int verbose_level);
   void compute_bisecants_and_conics(long int *arc6,
         int *&bisecants, int *&conics, int verbose_level);
      // bisecants[15 * 3]
      // conics[6 * 6]


};


// ##############################################################################
// buekenhout_metz.cpp
// ##############################################################################

//! Buekenhout-Metz unitals


class buekenhout_metz {
public:
   field_theory::finite_field *FQ, *Fq;
   int q;
   int Q;

   field_theory::subfield_structure *SubS;

   int f_classical;
   int f_Uab;
   int parameter_a;
   int parameter_b;
```

```
projective_space *P2; // PG(2,q^2), where the unital lives
projective_space *P3; // PG(3,q), where the ovoid lives

int *v; // [3]
int *w1; // [6]
int *w2; // [6]
int *w3; // [6]
int *w4; // [6]
int *w5; // [6]

//int *components;
//int *embedding;
//int *pair_embedding;

long int *ovoid;
long int *U;
int sz;
int alpha, t0, t1, T0, T1;
long int theta_3;
int minus_t0, sz_ovoid;
int e1, one_1, one_2;


// compute_the_design:
long int *secant_lines;
int nb_secant_lines;
long int *tangent_lines;
int nb_tangent_lines;
long int *Intersection_sets;
int *Design_blocks;
long int *block;
int block_size;
int *idx_in_unital;
int *idx_in_secants;
int *tangent_line_at_point;
int *point_of_tangency;
int *f_is_tangent_line;
int *f_is_Baer;


// the block that we choose:
int nb_good_points;
int *good_points; // = q + 1


buekenhout_metz();
~buekenhout_metz();
```

```
   void buekenhout_metz_init(
         field_theory::finite_field *Fq,
         field_theory::finite_field *FQ,
      int f_Uab, int a, int b,
      int f_classical, int verbose_level);
   void init_ovoid(int verbose_level);
   void init_ovoid_Uab_even(int a, int b, int verbose_level);
   void create_unital(int verbose_level);
   void create_unital_tex(int verbose_level);
   void create_unital_Uab_tex(int verbose_level);
   void compute_the_design(int verbose_level);
   void write_unital_to_file();
   void get_name(std::string &name);

};




// ##############################################################################
// decomposition.cpp
// ##############################################################################


//! decomposition of an incidence matrix


class decomposition {

public:

   int nb_points;
   int nb_blocks;
   int *Inc;
   incidence_structure *I;
   data_structures::partitionstack *Stack;

   int f_has_decomposition;
   int *row_classes;
   int *row_class_inv;
   int nb_row_classes;
   int *col_classes;
   int *col_class_inv;
   int nb_col_classes;
   int f_has_row_scheme;
   int *row_scheme;
   int f_has_col_scheme;
```

```
   int *col_scheme;



   decomposition();
   ~decomposition();
   void init_inc_and_stack(incidence_structure *Inc,
         data_structures::partitionstack *Stack,
      int verbose_level);
   void init_incidence_matrix(int m, int n, int *M,
      int verbose_level);
      // copies the incidence matrix
   void setup_default_partition(int verbose_level);
   void compute_TDO(int max_depth, int verbose_level);
   void print_row_decomposition_tex(std::ostream &ost,
      int f_enter_math, int f_print_subscripts,
      int verbose_level);
   void print_column_decomposition_tex(std::ostream &ost,
      int f_enter_math, int f_print_subscripts,
      int verbose_level);
   void get_row_scheme(int verbose_level);
   void get_col_scheme(int verbose_level);

};


// ##############################################################################
// desarguesian_spread.cpp
// ##############################################################################


//! desarguesian spread



class desarguesian_spread {
public:
   int n;
   int m;
   int s;
   int q;
   int Q;
   field_theory::finite_field *Fq;
   field_theory::finite_field *FQ;
   field_theory::subfield_structure *SubS;
   grassmann *Gr;
```

```
   int N;
      // = number of points in PG(m - 1, Q)

   int nb_points;
      // = number of points in PG(n - 1, q)

   int nb_points_per_spread_element;
      // = number of points in PG(s - 1, q)

   int spread_element_size;
      // = s * n

   int *Spread_elements;
      // [N * spread_element_size]

   long int *Rk;
      // [N]

   int *List_of_points;
      // [N * nb_points_per_spread_element]

   desarguesian_spread();
   ~desarguesian_spread();
   void init(int n, int m, int s,
         field_theory::subfield_structure *SubS,
      int verbose_level);
   void calculate_spread_elements(int verbose_level);
   void compute_intersection_type(int k, int *subspace,
      int *intersection_dimensions, int verbose_level);
   // intersection_dimensions[h]
   void compute_shadow(int *Basis, int basis_sz,
      int *is_in_shadow, int verbose_level);
   void compute_linear_set(int *Basis, int basis_sz,
      long int *&the_linear_set, int &the_linear_set_sz,
      int verbose_level);
   void print_spread_element_table_tex(std::ostream &ost);
   void print_spread_elements_tex(std::ostream &ost);
   void print_linear_set_tex(long int *set, int sz);
   void print_linear_set_element_tex(long int a, int sz);
   void create_latex_report(int verbose_level);
   void report(std::ostream &ost, int verbose_level);

};

// ##############################################################################
// flag.cpp
// ##############################################################################
```

```
//! a maximal chain of subspaces


class flag {
public:
   field_theory::finite_field *F;
   grassmann *Gr;
   int n;
   int s0, s1, s2;
   int k, K;
   int *type;
   int type_len;
   int idx;
   int N0, N, N1;
   flag *Flag;


   int *M; // [K * n]
   int *M_Gauss; // [K * n] the echelon form (RREF)
   int *transform; // [K * K] the transformation matrix, used as s2 * s2
   int *base_cols; // [n] base_cols for the matrix M_Gauss
   int *M1; // [n * n]
   int *M2; // [n * n]
   int *M3; // [n * n]

   flag();
   ~flag();
   void init(
         int n, int *type, int type_len,
         field_theory::finite_field *F,
      int verbose_level);
   void init_recursion(
         int n, int *type, int type_len, int idx,
         field_theory::finite_field *F, int verbose_level);
   void unrank(long int rk, int *subspace, int verbose_level);
   void unrank_recursion(long int rk, int *subspace, int verbose_level);
   long int rank(int *subspace, int verbose_level);
   long int rank_recursion(int *subspace, int *big_space, int verbose_level);
};


// ##############################################################################
// geometric_object_create.cpp
// ##############################################################################
```

```
//! to create a geometric object from a description using class geometric_object_
description



class geometric_object_create {

public:
   geometric_object_description *Descr;

   int nb_pts;
   long int *Pts;

   std::string label_txt;
   std::string label_tex;


   geometric_object_create();
   ~geometric_object_create();
   void init(geometric_object_description *Descr,
         projective_space *P, int verbose_level);
   void create_elliptic_quadric_ovoid(
         projective_space *P,
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
      int verbose_level);
   void create_ovoid_ST(
         projective_space *P,
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
      int verbose_level);
   void create_cuspidal_cubic(
         projective_space *P,
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
      int verbose_level);
   void create_twisted_cubic(
         projective_space *P,
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
      int verbose_level);
   void create_elliptic_curve(
         projective_space *P,
```

```
      int elliptic_curve_b, int elliptic_curve_c,
      std::string &label_txt,
      std::string &label_tex,
      int &nb_pts, long int *&Pts,
      int verbose_level);
   void create_unital_XXq_YZq_ZYq(
         projective_space *P,
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
      int verbose_level);
   void create_whole_space(
         projective_space *P,
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
         int verbose_level);
   void create_hyperplane(
         projective_space *P,
      int pt,
      std::string &label_txt,
      std::string &label_tex,
      int &nb_pts, long int *&Pts,
      int verbose_level);
   void create_Baer_substructure(
         projective_space *P,
      long int *&Pts, int &nb_pts,
      std::string &label_txt,
      std::string &label_tex,
      int verbose_level);
   // assumes we are in PG(n,Q) where Q = q^2
   void create_unital_XXq_YZq_ZYq_brute_force(
         projective_space *P,
         long int *U, int &sz, int verbose_level);

};



// ##############################################################################
// geometric_object_description.cpp
// ##############################################################################


//! to create a geometric object encoded as a set using a description from the co
mmand line
```

```
class geometric_object_description {

public:

   projective_space *P;

   int f_subiaco_oval;
   int f_short;
   int f_subiaco_hyperoval;
   int f_adelaide_hyperoval;

   int f_hyperoval;
   int f_translation;
   int translation_exponent;
   int f_Segre;
   int f_Payne;
   int f_Cherowitzo;
   int f_OKeefe_Penttila;

   int f_BLT_database;
   int BLT_database_k;
   int f_BLT_database_embedded;
   int BLT_database_embedded_k;

   int f_elliptic_quadric_ovoid;
   int f_ovoid_ST;

   int f_Baer_substructure;

   int f_orthogonal;
   int orthogonal_epsilon;

   int f_hermitian;

   int f_cuspidal_cubic; // twisted cubic in PG(2,q)
   int f_twisted_cubic; // twisted cubic in PG(3,q)

   int f_elliptic_curve;
   int elliptic_curve_b;
   int elliptic_curve_c;

   int f_ttp_code;
   int f_ttp_construction_A;
   int f_ttp_hyperoval;
   int f_ttp_construction_B;
```

```
int f_unital_XXq_YZq_ZYq;

int f_desarguesian_line_spread_in_PG_3_q;
int f_embedded_in_PG_4_q;

int f_Buekenhout_Metz;
int f_classical;
int f_Uab;
int parameter_a;
int parameter_b;

int f_whole_space;
int f_hyperplane;
int pt;

int f_segre_variety;
int segre_variety_a;
int segre_variety_b;

int f_Maruta_Hamada_arc;

int f_projective_variety;
std::string projective_variety_ring_label;
std::string variety_label_txt;
std::string variety_label_tex;
std::string variety_coeffs;


int f_intersection_of_zariski_open_sets;
std::string intersection_of_zariski_open_sets_ring_label;
std::vector<std::string> Variety_coeffs;

int f_number_of_conditions_satisfied;
std::string number_of_conditions_satisfied_ring_label;
std::string number_of_conditions_satisfied_fname;


int f_projective_curve;
std::string projective_curve_ring_label;
std::string curve_label_txt;
std::string curve_label_tex;
std::string curve_coeffs;

int f_set;
std::string set_text;
```

```
   geometric_object_description();
   ~geometric_object_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};





// ##############################################################################
// geometry_global.cpp
// ##############################################################################


//! various functions related to geometries



class geometry_global {
public:

   geometry_global();
   ~geometry_global();
   long int nb_PG_elements(int n, int q);
      // $\frac{q^{n+1} - 1}{q-1} = \sum_{i=0}^{n} q^i $
   long int nb_PG_elements_not_in_subspace(int n, int m, int q);
   long int nb_AG_elements(int n, int q);
   long int nb_affine_lines(int n, int q);
   long int AG_element_rank(
         int q, int *v, int stride, int len);
   void AG_element_unrank(
         int q, int *v, int stride, int len, long int a);
   int AG_element_next(
         int q, int *v, int stride, int len);
   void AG_element_rank_longinteger(
         int q, int *v, int stride, int len,
         ring_theory::longinteger_object &a);
   void AG_element_unrank_longinteger(
         int q, int *v, int stride, int len,
         ring_theory::longinteger_object &a);
   int PG_element_modified_is_in_subspace(
         int n, int m, int *v);
```

```
int test_if_arc(field_theory::finite_field *Fq,
      int *pt_coords, int *set,
      int set_sz, int k, int verbose_level);
void create_Buekenhout_Metz(
      field_theory::finite_field *Fq,
      field_theory::finite_field *FQ,
   int f_classical, int f_Uab, int parameter_a, int parameter_b,
   std::string &fname, int &nb_pts, long int *&Pts,
   int verbose_level);
long int count_Sbar(int n, int q);
long int count_S(int n, int q);
long int count_N1(int n, int q);
long int count_T1(int epsilon, int n, int q);
long int count_T2(int n, int q);
long int nb_pts_Qepsilon(int epsilon, int k, int q);
// number of singular points on Q^epsilon(k,q)
int dimension_given_Witt_index(int epsilon, int n);
int Witt_index(int epsilon, int k);
long int nb_pts_Q(int k, int q);
// number of singular points on Q(k,q)
long int nb_pts_Qplus(int k, int q);
// number of singular points on Q^+(k,q)
long int nb_pts_Qminus(int k, int q);
// number of singular points on Q^-(k,q)
long int nb_pts_S(int n, int q);
long int nb_pts_N(int n, int q);
long int nb_pts_N1(int n, int q);
long int nb_pts_Sbar(int n, int q);
long int nb_pts_Nbar(int n, int q);
//void test_Orthogonal(int epsilon, int k, int q);
//void test_orthogonal(int n, int q);
int &TDO_upper_bound(int i, int j);
int &TDO_upper_bound_internal(int i, int j);
int &TDO_upper_bound_source(int i, int j);
int braun_test_single_type(int v, int k, int ak);
int braun_test_upper_bound(int v, int k);
void TDO_refine_init_upper_bounds(int v_max);
void TDO_refine_extend_upper_bounds(int new_v_max);
int braun_test_on_line_type(int v, int *type);
int &maxfit(int i, int j);
int &maxfit_internal(int i, int j);
void maxfit_table_init(int v_max);
void maxfit_table_reallocate(int v_max);
void maxfit_table_compute();
int packing_number_via_maxfit(int n, int k);
void do_inverse_isomorphism_klein_quadric(
      field_theory::finite_field *F,
```

```
      std::string &inverse_isomorphism_klein_quadric_matrix_A6,
      int verbose_level);
// creates klein_correspondence and orthogonal_geometry::orthogonal objects
void do_rank_points_in_PG(
      field_theory::finite_field *F,
      std::string &label,
      int verbose_level);
void do_unrank_points_in_PG(
      field_theory::finite_field *F,
      int n,
      std::string &text,
      int verbose_level);
void do_intersection_of_two_lines(field_theory::finite_field *F,
      std::string &line_1_basis,
      std::string &line_2_basis,
      int f_normalize_from_the_left, int f_normalize_from_the_right,
      int verbose_level);
void do_transversal(field_theory::finite_field *F,
      std::string &line_1_basis,
      std::string &line_2_basis,
      std::string &point,
      int f_normalize_from_the_left, int f_normalize_from_the_right,
      int verbose_level);
void do_cheat_sheet_hermitian(
      field_theory::finite_field *F,
      int projective_dimension,
      int verbose_level);
// creates a hermitian object
void do_create_desarguesian_spread(
      field_theory::finite_field *FQ,
      field_theory::finite_field *Fq,
      int m,
      int verbose_level);
// creates field_theory::subfield_structure and desarguesian_spread objects
void create_decomposition_of_projective_plane(
      std::string &fname_base,
      projective_space *P,
      long int *points, int nb_points,
      long int *lines, int nb_lines,
      int verbose_level);
// creates incidence_structure and data_structures::partitionstack objects
void create_BLT_point(field_theory::finite_field *F,
      int *v5, int a, int b, int c, int verbose_level);
   // creates the point (-b/2,-c,a,-(b^2/4-ac),1)
   // check if it satisfies x_0^2 + x_1x_2 + x_3x_4:
   // b^2/4 + (-c)*a + -(b^2/4-ac)
   // = b^2/4 -ac -b^2/4 + ac = 0
```

```
   int nonconical_six_arc_get_nb_Eckardt_points(
         projective_space *P2,
         long int *Arc6, int verbose_level);
   algebraic_geometry::eckardt_point_info *compute_eckardt_point_info(
         projective_space *P2,
      long int *arc6,
      int verbose_level);
   int test_nb_Eckardt_points(
         projective_space *P2,
         long int *S, int len, int pt, int nb_E, int verbose_level);
   void rearrange_arc_for_lifting(
         long int *Arc6,
         long int P1, long int P2, int partition_rk, long int *arc,
         int verbose_level);
   void find_two_lines_for_arc_lifting(
         projective_space *P,
         long int P1, long int P2, long int &line1, long int &line2,
         int verbose_level);
   void hyperplane_lifting_with_two_lines_fixed(
         projective_space *P,
         int *A3, int f_semilinear, int frobenius,
         long int line1, long int line2,
         int *A4,
         int verbose_level);
   void hyperplane_lifting_with_two_lines_moved(
         projective_space *P,
         long int line1_from, long int line1_to,
         long int line2_from, long int line2_to,
         int *A4,
         int verbose_level);
#if 0
   void andre_preimage(
         projective_space *P2, projective_space *P4,
      long int *set2, int sz2, long int *set4, int &sz4,
      int verbose_level);
#endif
   void find_secant_lines(
         projective_space *P,
         long int *set, int set_size,
         long int *lines, int &nb_lines, int max_lines,
         int verbose_level);
   void find_lines_which_are_contained(
         projective_space *P,
         std::vector<long int> &Points,
         std::vector<long int> &Lines,
         int verbose_level);
   void do_move_two_lines_in_hyperplane_stabilizer(
```

```
      projective_space *P3,
      long int line1_from, long int line2_from,
      long int line1_to, long int line2_to, int verbose_level);
void do_move_two_lines_in_hyperplane_stabilizer_text(
      projective_space *P3,
      std::string &line1_from_text, std::string &line2_from_text,
      std::string &line1_to_text, std::string &line2_to_text,
      int verbose_level);
void make_restricted_incidence_matrix(
      geometry::projective_space *P,
      int type_i, int type_j,
      std::string &row_objects,
      std::string &col_objects,
      std::string &file_name,
      int verbose_level);
void plane_intersection_type_of_klein_image(
      geometry::projective_space *P,
      std::string &input,
      int threshold,
      int verbose_level);
// creates a projective_space object P5
void conic_type(
      geometry::projective_space *P,
      int threshold,
      std::string &set_text,
      int verbose_level);
void conic_type2(
      geometry::projective_space *P,
      long int *Pts, int nb_pts, int threshold,
      int verbose_level);
void do_rank_lines_in_PG(
      geometry::projective_space *P,
      std::string &label,
      int verbose_level);
void do_unrank_lines_in_PG(
      geometry::projective_space *P,
      std::string &label,
      int verbose_level);
void do_cone_over(int n,
      field_theory::finite_field *F,
   long int *set_in, int set_size_in,
   long int *&set_out, int &set_size_out,
   int verbose_level);
void do_blocking_set_family_3(int n,
      field_theory::finite_field *F,
   long int *set_in, int set_size,
   long int *&the_set_out, int &set_size_out,
```

```
      int verbose_level);
   // creates projective_space PG(2,q)
   void create_orthogonal(
         field_theory::finite_field *F,
         int epsilon, int n,
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
      int verbose_level);
   // creates a quadratic form
   void create_hermitian(
         field_theory::finite_field *F,
         int n,
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
      int verbose_level);
   // creates hermitian
   void create_ttp_code(
         field_theory::finite_field *FQ,
         field_theory::finite_field *Fq_subfield,
      int f_construction_A, int f_hyperoval, int f_construction_B,
      std::string &fname, int &nb_pts, long int *&Pts,
      int verbose_level);
   // creates a projective_space
   void create_segre_variety(
         field_theory::finite_field *F,
         int a, int b,
         std::string &label_txt,
         std::string &label_tex,
         int &nb_pts, long int *&Pts,
      int verbose_level);
   // The Segre map goes from PG(a,q) cross PG(b,q) to PG((a+1)*(b+1)-1,q)
#if 0
   void do_andre(
         field_theory::finite_field *FQ,
         field_theory::finite_field *Fq,
         long int *the_set_in, int set_size_in,
         long int *&the_set_out, int &set_size_out,
      int verbose_level);
   // creates PG(2,Q) and PG(4,q)
   // this functions is not called from anywhere right now
   // it needs a pair of finite fields
#endif
   void do_embed_orthogonal(
         field_theory::finite_field *F,
      int epsilon, int n,
```

```
      long int *set_in, long int *&set_out, int set_size,
      int verbose_level);
   // creates a quadratic_form object
   void do_embed_points(
         field_theory::finite_field *F,
         int n,
         long int *set_in, long int *&set_out, int set_size,
      int verbose_level);
   void print_set_in_affine_plane(
         field_theory::finite_field *F,
         int len, long int *S);
   void simeon(
         field_theory::finite_field *F,
         int n, int len, long int *S, int s, int verbose_level);
   void wedge_to_klein(
         field_theory::finite_field *F,
         int *W, int *K);
   void klein_to_wedge(
         field_theory::finite_field *F,
         int *K, int *W);
   void isomorphism_to_special_orthogonal(
         field_theory::finite_field *F,
         int *A4, int *A6, int verbose_level);
   void minimal_orbit_rep_under_stabilizer_of_frame_characteristic_two(
         field_theory::finite_field *F,
         int x, int y,
         int &a, int &b, int verbose_level);
   int evaluate_Fermat_cubic(
         field_theory::finite_field *F,
         int *v);

};


// ##############################################################################
// grassmann.cpp
// ##############################################################################

//! to rank and unrank subspaces of a fixed dimension in F_q^n


class grassmann {
public:
   int n, k, q;
   ring_theory::longinteger_object *nCkq; // n choose k q-analog
   field_theory::finite_field *F;
   int *base_cols;
```

```
int *coset;
int *M; // [n * n], this used to be [k * n]
   // but now we allow for embedded subspaces.
int *M2; // [n * n], used in dual_spread
int *v; // [n], for points_covered
int *w; // [n], for points_covered
grassmann *G;

grassmann();
~grassmann();
void init(int n, int k,
      field_theory::finite_field *F, int verbose_level);
long int nb_of_subspaces(int verbose_level);
void print_single_generator_matrix_tex(std::ostream &ost, long int a);
void print_single_generator_matrix_tex_numerical(
      std::ostream &ost, long int a);
void print_set(long int *v, int len);
void print_set_tex(std::ostream &ost,
      long int *v, int len, int verbose_level);
void print_set_tex_with_perp(
      std::ostream &ost, long int *v, int len);
int nb_points_covered(int verbose_level);
void points_covered(long int *the_points, int verbose_level);
void unrank_lint_here(int *Mtx, long int rk, int verbose_level);
long int rank_lint_here(int *Mtx, int verbose_level);
void unrank_embedded_subspace_lint(
      long int rk, int verbose_level);
long int rank_embedded_subspace_lint(int verbose_level);
void unrank_embedded_subspace_lint_here(
      int *Basis, long int rk, int verbose_level);
void unrank_lint(long int rk, int verbose_level);
long int rank_lint(int verbose_level);
void unrank_longinteger_here(int *Mtx,
      ring_theory::longinteger_object &rk,
   int verbose_level);
void rank_longinteger_here(int *Mtx,
      ring_theory::longinteger_object &rk,
   int verbose_level);
void unrank_longinteger(
      ring_theory::longinteger_object &rk, int verbose_level);
void rank_longinteger(
      ring_theory::longinteger_object &r, int verbose_level);
void print();
int dimension_of_join(
      long int rk1, long int rk2, int verbose_level);
void unrank_lint_here_and_extend_basis(int *Mtx, long int rk,
   int verbose_level);
```

```
   // Mtx must be n x n
void unrank_lint_here_and_compute_perp(int *Mtx, long int rk,
   int verbose_level);
   // Mtx must be n x n
void line_regulus_in_PG_3_q(long int *&regulus,
   int &regulus_size, int f_opposite, int verbose_level);
   // the equation of the hyperboloid is x_0x_3-x_1x_2 = 0
void compute_dual_line_idx(int *&dual_line_idx,
      int *&self_dual_lines, int &nb_self_dual_lines,
      int verbose_level);
void compute_dual_spread(int *spread, int *dual_spread,
   int spread_size, int verbose_level);
void latex_matrix(std::ostream &ost, int *p);
void latex_matrix_numerical(std::ostream &ost, int *p);
void create_Schlaefli_graph(
      int *&Adj, int &sz, int verbose_level);
long int make_special_element_zero(int verbose_level);
long int make_special_element_one(int verbose_level);
long int make_special_element_infinity(int verbose_level);
void make_identity_front(int *M, int verbose_level);
void make_identity_back(int *M, int verbose_level);
void copy_matrix_back(int *A, int *M, int verbose_level);
void extract_matrix_from_back(
      int *A, int *M, int verbose_level);
void make_spread_from_spread_set(
      long int *Spread_set, int sz,
      long int *&Spread, int &spread_sz,
      int verbose_level);
void make_spread_set_from_spread(
      long int *Spread, int spread_sz,
      int *&Spread_set, int &sz,
      int verbose_level);
void make_partition(long int *Spread, int spread_sz,
      long int *&Part, int &s, int verbose_level);
void make_spread_element(
      int *Spread_element, int *A, int verbose_level);
void cheat_sheet_subspaces(std::ostream &f, int verbose_level);
void Pluecker_coordinates(
   int line_rk, int *v6, int verbose_level);
void do_pluecker_reverse(std::ostream &ost,
      int nb_k_subspaces, int verbose_level);
void create_latex_report(int verbose_level);
void klein_correspondence(
      projective_space *P3,
      //projective_space *P5,
   long int *set_in, int set_size,
   long int *set_out, int verbose_level);
```

```
      // Computes the Pluecker coordinates
      // for a set of lines in PG(3,q) in the following order:
      // (x_1,x_2,x_3,x_4,x_5,x_6) =
      // (Pluecker_12, Pluecker_34, Pluecker_13, Pluecker_42,
      //  Pluecker_14, Pluecker_23)
      // satisfying the quadratic form
      // x_1x_2 + x_3x_4 + x_5x_6 = 0
   void klein_correspondence_special_model(
         projective_space *P3,
         //projective_space *P5,
      long int *table, int verbose_level);
   void plane_intersection_type_of_klein_image(
         geometry::projective_space *P3,
         geometry::projective_space *P5,
         long int *data, int size, int threshold,
         intersection_type *&Int_type,
         int verbose_level);
   void get_spread_matrices(int *G, int *H,
         long int *data, int verbose_level);
   // assuming we are in PG(3,q)

};


// ##############################################################################
// grassmann_embedded.cpp
// ##############################################################################

//! subspaces with a fixed embedding


class grassmann_embedded {
public:
   int big_n, n, k, q;
   field_theory::finite_field *F;
   grassmann *G; // only a reference, not freed
   int *M; // [n * big_n] the original matrix
   int *M_Gauss; // [n * big_n] the echelon form (RREF)
   int *transform; // [n * n] the transformation matrix
   int *base_cols; // [n] base_cols for the matrix M_Gauss
   int *embedding;
      // [big_n - n], the columns which are not
      // base_cols in increasing order
   int *Tmp1; // [big_n]
   int *Tmp2; // [big_n]
   int *Tmp3; // [big_n]
   int *tmp_M1; // [n * n]
```

```
   int *tmp_M2; // [n * n]
   long int degree; // q_binomial n choose k


   grassmann_embedded();
   ~grassmann_embedded();
   void init(int big_n, int n, grassmann *G, int *M, int verbose_level);
      // M is n x big_n
   void unrank_embedded_lint(int *subspace_basis_with_embedding,
      long int rk, int verbose_level);
      // subspace_basis_with_embedding is n x big_n
   long int rank_embedded_lint(int *subspace_basis, int verbose_level);
      // subspace_basis is n x big_n,
      // only the first k x big_n entries are used
   void unrank_lint(int *subspace_basis, long int rk, int verbose_level);
      // subspace_basis is k x big_n
   long int rank_lint(int *subspace_basis, int verbose_level);
      // subspace_basis is k x big_n
};

// ##############################################################################
// hermitian.cpp
// ##############################################################################

//! hermitian space


class hermitian {

public:

   // The hermitian form is \sum_{i=0}^{k-1} X_i^{q+1}

   field_theory::finite_field *F; // only a reference, not to be freed
   int Q;
   int q;
   int k; // nb_vars

   int *cnt_N; // [k + 1]
   int *cnt_N1; // [k + 1]
   int *cnt_S; // [k + 1]
   int *cnt_Sbar; // [k + 1]

   int *norm_one_elements; // [q + 1]
   int *index_of_norm_one_element; // [Q]
   int alpha; // a primitive element for GF(Q), namely F->p
   int beta; // alpha^(q+1), a primitive element for GF(q)
```

```
   int *log_beta; // [Q]
   int *beta_power; // [q - 1]
      // beta_power[i] = beta to the power i = j
      // log_beta[j] = i

   hermitian();
   ~hermitian();
   void init(field_theory::finite_field *F,
         int nb_vars, int verbose_level);
   int nb_points();
   void unrank_point(int *v, int rk);
   int rank_point(int *v);
   void list_of_points_embedded_in_PG(
         long int *&Pts, int &nb_pts,
      int verbose_level);
   void list_all_N(int verbose_level);
   void list_all_N1(int verbose_level);
   void list_all_S(int verbose_level);
   void list_all_Sbar(int verbose_level);
   int evaluate_hermitian_form(int *v, int len);
   void N_unrank(int *v, int len, int rk, int verbose_level);
   int N_rank(int *v, int len, int verbose_level);
   void N1_unrank(int *v, int len, int rk, int verbose_level);
   int N1_rank(int *v, int len, int verbose_level);
   void S_unrank(int *v, int len, int rk, int verbose_level);
   int S_rank(int *v, int len, int verbose_level);
   void Sbar_unrank(int *v, int len, int rk, int verbose_level);
   int Sbar_rank(int *v, int len, int verbose_level);
   void create_latex_report(int verbose_level);
   void report(std::ostream &ost, int verbose_level);
   void report_points(std::ostream &ost, int verbose_level);

};

// ##############################################################################
// hjelmslev.cpp
// ##############################################################################

//! Hjelmslev geometry


class hjelmslev {
public:
   int n, k, q;
   int n_choose_k_p;
   ring_theory::finite_ring *R; // do not free
   grassmann *G;
```

```
   int *v;
   int *Mtx;
   int *base_cols;

   hjelmslev();
   ~hjelmslev();
   void init(ring_theory::finite_ring *R,
         int n, int k, int verbose_level);
   long int number_of_submodules();
   void unrank_lint(int *M, long int rk, int verbose_level);
   long int rank_lint(int *M, int verbose_level);
};

// ##############################################################################
// incidence_structure.cpp
// ##############################################################################

#define INCIDENCE_STRUCTURE_REALIZATION_BY_MATRIX 1
#define INCIDENCE_STRUCTURE_REALIZATION_BY_ORTHOGONAL 2
#define INCIDENCE_STRUCTURE_REALIZATION_BY_HJELMSLEV 3
#define INCIDENCE_STRUCTURE_REALIZATION_BY_PROJECTIVE_SPACE 4

//! interface for various incidence geometries


class incidence_structure {
   public:

   std::string label;


   int nb_rows;
   int nb_cols;


   int f_rowsums_constant;
   int f_colsums_constant;
   int r;
   int k;
   int *nb_lines_on_point;
   int *nb_points_on_line;
   int max_r;
   int min_r;
   int max_k;
   int min_k;
   int *lines_on_point; // [nb_rows * max_r]
   int *points_on_line; // [nb_cols * max_k]
```

```
int realization_type;
   // INCIDENCE_STRUCTURE_REALIZATION_BY_MATRIX
   // INCIDENCE_STRUCTURE_REALIZATION_BY_ORTHOGONAL

int *M;
orthogonal_geometry::orthogonal *O;
hjelmslev *H;
projective_space *P;


incidence_structure();
~incidence_structure();
void check_point_pairs(int verbose_level);
int lines_through_two_points(int *lines, int p1, int p2,
   int verbose_level);
void init_projective_space(
      projective_space *P, int verbose_level);
void init_hjelmslev(
      hjelmslev *H, int verbose_level);
void init_orthogonal(
      orthogonal_geometry::orthogonal *O,
      int verbose_level);
void init_by_incidences(int m, int n, int nb_inc, int *X,
   int verbose_level);
void init_by_R_and_X(int m, int n, int *R, int *X, int max_r,
   int verbose_level);
void init_by_set_of_sets(
      data_structures::set_of_sets *SoS,
      int verbose_level);
void init_by_matrix(int m, int n, int *M, int verbose_level);
void init_by_matrix_as_bitmatrix(
      int m, int n,
      data_structures::bitmatrix *Bitmatrix,
      int verbose_level);
void init_by_matrix2(int verbose_level);
int nb_points();
int nb_lines();
int get_ij(int i, int j);
int get_lines_on_point(int *data, int i, int verbose_level);
int get_points_on_line(int *data, int j, int verbose_level);
int get_nb_inc();
void save_inc_file(char *fname);
void save_row_by_row_file(char *fname);
void print(std::ostream &ost);
void compute_TDO_safe_first(
      data_structures::partitionstack &PStack,
```

```
   int depth, int &step, int &f_refine,
   int &f_refine_prev, int verbose_level);
int compute_TDO_safe_next(
      data_structures::partitionstack &PStack,
   int depth, int &step, int &f_refine,
   int &f_refine_prev, int verbose_level);
   // returns TRUE when we are done, FALSE otherwise
void compute_TDO_safe(
      data_structures::partitionstack &PStack,
   int depth, int verbose_level);
int compute_TDO(
      data_structures::partitionstack &PStack,
      int ht0, int depth,
   int verbose_level);
int compute_TDO_step(
      data_structures::partitionstack &PStack, int ht0,
   int verbose_level);
void get_partition(
      data_structures::partitionstack &PStack,
   int *row_classes, int *row_class_idx, int &nb_row_classes,
   int *col_classes, int *col_class_idx, int &nb_col_classes);
int refine_column_partition_safe(
      data_structures::partitionstack &PStack,
   int verbose_level);
int refine_row_partition_safe(
      data_structures::partitionstack &PStack,
   int verbose_level);
int refine_column_partition(
      data_structures::partitionstack &PStack, int ht0,
   int verbose_level);
int refine_row_partition(
      data_structures::partitionstack &PStack, int ht0,
   int verbose_level);
void print_row_tactical_decomposition_scheme_incidences_tex(
      data_structures::partitionstack &PStack,
   std::ostream &ost, int f_enter_math_mode,
   int *row_classes, int *row_class_inv, int nb_row_classes,
   int *col_classes, int *col_class_inv, int nb_col_classes,
   int f_local_coordinates, int verbose_level);
void print_col_tactical_decomposition_scheme_incidences_tex(
      data_structures::partitionstack &PStack,
   std::ostream &ost, int f_enter_math_mode,
   int *row_classes, int *row_class_inv, int nb_row_classes,
   int *col_classes, int *col_class_inv, int nb_col_classes,
   int f_local_coordinates, int verbose_level);
void get_incidences_by_row_scheme(
      data_structures::partitionstack &PStack,
```

```
   int *row_classes, int *row_class_inv, int nb_row_classes,
   int *col_classes, int *col_class_inv, int nb_col_classes,
   int row_class_idx, int col_class_idx,
   int rij, int *&incidences, int verbose_level);
void get_incidences_by_col_scheme(
      data_structures::partitionstack &PStack,
   int *row_classes, int *row_class_inv, int nb_row_classes,
   int *col_classes, int *col_class_inv, int nb_col_classes,
   int row_class_idx, int col_class_idx,
   int kij, int *&incidences, int verbose_level);
void get_row_decomposition_scheme(
      data_structures::partitionstack &PStack,
   int *row_classes, int *row_class_inv, int nb_row_classes,
   int *col_classes, int *col_class_inv, int nb_col_classes,
   int *row_scheme, int verbose_level);
void get_row_decomposition_scheme_if_possible(
      data_structures::partitionstack &PStack,
   int *row_classes, int *row_class_inv, int nb_row_classes,
   int *col_classes, int *col_class_inv, int nb_col_classes,
   int *row_scheme, int verbose_level);
void get_col_decomposition_scheme(
      data_structures::partitionstack &PStack,
   int *row_classes, int *row_class_inv, int nb_row_classes,
   int *col_classes, int *col_class_inv, int nb_col_classes,
   int *col_scheme, int verbose_level);

void row_scheme_to_col_scheme(
      data_structures::partitionstack &PStack,
   int *row_classes, int *row_class_inv, int nb_row_classes,
   int *col_classes, int *col_class_inv, int nb_col_classes,
   int *row_scheme, int *col_scheme, int verbose_level);
void get_and_print_row_decomposition_scheme(
      data_structures::partitionstack &PStack,
   int f_list_incidences,
   int f_local_coordinates, int verbose_level);
void get_and_print_col_decomposition_scheme(
      data_structures::partitionstack &PStack,
   int f_list_incidences,
   int f_local_coordinates, int verbose_level);
void get_and_print_decomposition_schemes(
      data_structures::partitionstack &PStack);
void get_and_print_decomposition_schemes_tex(
      data_structures::partitionstack &PStack);
void get_and_print_tactical_decomposition_scheme_tex(
      std::ostream &ost, int f_enter_math,
      data_structures::partitionstack &PStack);
void get_scheme(
```

```
   int *&row_classes, int *&row_class_inv, int &nb_row_classes,
   int *&col_classes, int *&col_class_inv, int &nb_col_classes,
   int *&scheme, int f_row_scheme,
   data_structures::partitionstack &PStack);
void free_scheme(
   int *row_classes, int *row_class_inv,
   int *col_classes, int *col_class_inv,
   int *scheme);
void get_and_print_row_tactical_decomposition_scheme_tex(
      std::ostream &ost,
      int f_enter_math, int f_print_subscripts,
      data_structures::partitionstack &PStack);
void get_and_print_column_tactical_decomposition_scheme_tex(
      std::ostream &ost,
      int f_enter_math, int f_print_subscripts,
      data_structures::partitionstack &PStack);
void print_non_tactical_decomposition_scheme_tex(
      std::ostream &ost, int f_enter_math,
      data_structures::partitionstack &PStack);
void print_line(std::ostream &ost,
      data_structures::partitionstack &P,
   int row_cell, int i, int *col_classes, int nb_col_classes,
   int width, int f_labeled);
void print_column_labels(
      std::ostream &ost, data_structures::partitionstack &P,
   int *col_classes, int nb_col_classes, int width);
void print_hline(std::ostream &ost,
      data_structures::partitionstack &P,
   int *col_classes, int nb_col_classes,
   int width, int f_labeled);
void print_partitioned(std::ostream &ost,
      data_structures::partitionstack &P, int f_labeled);
void point_collinearity_graph(int *Adj, int verbose_level);
   // G[nb_points() * nb_points()]
void line_intersection_graph(int *Adj, int verbose_level);
   // G[nb_lines() * nb_lines()]
void latex_it(std::ostream &ost,
      data_structures::partitionstack &P);
void rearrange(int *&Vi, int &nb_V,
   int *&Bj, int &nb_B, int *&R, int *&X,
   data_structures::partitionstack &P);
void decomposition_print_tex(std::ostream &ost,
      data_structures::partitionstack &PStack,
      int f_row_tactical, int f_col_tactical,
   int f_detailed,
   int f_local_coordinates, int verbose_level);
void do_tdo_high_level(
```

```
      data_structures::partitionstack &S,
   int f_tdo_steps, int f_tdo_depth, int tdo_depth,
   int f_write_tdo_files, int f_pic,
   int f_include_tdo_scheme, int f_include_tdo_extra,
   int f_write_tdo_class_files,
   int verbose_level);
void compute_tdo(
      data_structures::partitionstack &S,
   int f_write_tdo_files,
   int f_pic,
   int f_include_tdo_scheme,
   int verbose_level);
void compute_tdo_stepwise(
      data_structures::partitionstack &S,
   int TDO_depth,
   int f_write_tdo_files,
   int f_pic,
   int f_include_tdo_scheme,
   int f_include_extra,
   int verbose_level);
void init_partitionstack_trivial(
      data_structures::partitionstack *S,
   int verbose_level);
void init_partitionstack(
      data_structures::partitionstack *S,
   int f_row_part, int nb_row_parts, int *row_parts,
   int f_col_part, int nb_col_parts, int *col_parts,
   int nb_distinguished_point_sets,
      int **distinguished_point_sets,
      int *distinguished_point_set_size,
   int nb_distinguished_line_sets,
      int **distinguished_line_sets,
      int *distinguished_line_set_size,
   int verbose_level);
void shrink_aut_generators(
   int nb_distinguished_point_sets,
   int nb_distinguished_line_sets,
   int Aut_counter, int *Aut, int *Base, int Base_length,
   int verbose_level);
void print_aut_generators(int Aut_counter, int *Aut,
   int Base_length, int *Base, int *Transversal_length);
void compute_extended_collinearity_graph(
   int *&Adj, int &v, int *&partition,
   int f_row_part, int nb_row_parts, int *row_parts,
   int f_col_part, int nb_col_parts, int *col_parts,
   int nb_distinguished_point_sets,
      int **distinguished_point_sets,
```

```
         int *distinguished_point_set_size,
      int nb_distinguished_line_sets,
         int **distinguished_line_sets,
         int *distinguished_line_set_size,
      int verbose_level);
      // side effect: the distinguished sets
      // will be sorted afterwards
   void compute_extended_matrix(
      int *&M, int &nb_rows, int &nb_cols,
      int &total, int *&partition,
      int f_row_part, int nb_row_parts, int *row_parts,
      int f_col_part, int nb_col_parts, int *col_parts,
      int nb_distinguished_point_sets,
      int **distinguished_point_sets,
      int *distinguished_point_set_size,
      int nb_distinguished_line_sets,
      int **distinguished_line_sets,
      int *distinguished_line_set_size,
      int verbose_level);
   data_structures::bitvector *encode_as_bitvector();
   incidence_structure *apply_canonical_labeling(
         long int *canonical_labeling, int verbose_level);
   void save_as_csv(std::string &fname_csv, int verbose_level);
   void init_large_set(
         long int *blocks,
         int N_points, int design_b, int design_k,
         int partition_class_size,
         int *&partition, int verbose_level);
};



// ##############################################################################
// intersection_type.cpp
// ##############################################################################



//! intersection type of a set of points with respect to subspaces of a certain d
imension


class intersection_type {

public:

   long int *set;
```

```
   int set_size;
   int threshold;

   projective_space *P;
   grassmann *Gr;

   ring_theory::longinteger_object *R;
   long int **Pts_on_plane;
   int *nb_pts_on_plane;
   int len;

   int *the_intersection_type;
   int highest_intersection_number;

   long int *Highest_weight_objects;
   int nb_highest_weight_objects;

   int *Intersection_sets;

   data_structures::int_matrix *M;

   intersection_type();
   ~intersection_type();
   void plane_intersection_type_slow(
      long int *set, int set_size, int threshold,
      projective_space *P,
      grassmann *Gr,
      int verbose_level);
   void compute_heighest_weight_objects(int verbose_level);


};



// ##############################################################################
// klein_correspondence.cpp
// ##############################################################################


//! the Klein correspondence between lines in PG(3,q) and points on the Klein qua
dric


class klein_correspondence {
public:
```

```
// Pluecker coordinates of a line in PG(3,q) are:
// (p_1,p_2,p_3,p_4,p_5,p_6) =
// (Pluecker_12, Pluecker_34, Pluecker_13,
//    Pluecker_42, Pluecker_14, Pluecker_23)
// satisfying the quadratic form p_1p_2 + p_3p_4 + p_5p_6 = 0
// Here, the line is given as the rowspan of the matrix
// x_1 x_2 x_3 x_4
// y_1 y_2 y_3 y_4
// and
// Pluecker_ij is the determinant of the 2 x 2 submatrix
// formed by restricting the generator matrix to columns i and j.

projective_space *P3;
projective_space *P5;
orthogonal_geometry::orthogonal *O;
field_theory::finite_field *F;
int q;
long int nb_Pts; // number of points on the Klein quadric
long int nb_pts_PG; // number of points in PG(5,q)

grassmann *Gr63;
grassmann *Gr62;

long int nb_lines_orthogonal;

int *Form; // [d * d]

//long int *Line_to_point_on_quadric; // [P3->N_lines]
//long int *Point_on_quadric_to_line; // [P3->N_lines]

// too much storage:
//long int *Point_on_quadric_embedded_in_P5; // [P3->N_lines]
//int *coordinates_of_quadric_points; // [P3->N_lines * d]
//int *Pt_rk; // [P3->N_lines]

//int *Pt_idx; // [nb_pts_PG] too memory intense

klein_correspondence();
~klein_correspondence();
void init(
      field_theory::finite_field *F,
      orthogonal_geometry::orthogonal *O,
      int verbose_level);
// opens two projective_space objects P3 and P5
void plane_intersections(long int *lines_in_PG3, int nb_lines,
      ring_theory::longinteger_object *&R,
   long int **&Pts_on_plane,
```

```
      int *&nb_pts_on_plane,
      int &nb_planes,
      int verbose_level);
   long int point_on_quadric_embedded_in_P5(long int pt);
   long int line_to_point_on_quadric(long int line_rk, int verbose_level);
   void line_to_Pluecker(long int line_rk, int *v6, int verbose_level);
   long int point_on_quadric_to_line(long int point_rk, int verbose_level);
   void Pluecker_to_line(int *v6, int *basis_line, int verbose_level);
   long int Pluecker_to_line_rk(int *v6, int verbose_level);
   void exterior_square_to_line(
         int *v, int *basis_line,
         int verbose_level);
   void compute_external_lines(
         std::vector<long int> &External_lines,
         int verbose_level);
   void identify_external_lines_and_spreads(
         spread_tables *T,
         std::vector<long int> &External_lines,
         long int *&spread_to_external_line_idx,
         long int *&external_line_to_spread,
         int verbose_level);
   // spread_to_external_line_idx[i] is index into External_lines
   // corresponding to regular spread i
   // external_line_to_spread[i] is the index of the
   // regular spread of PG(3,q) in table T associated with
   // External_lines[i]
   void reverse_isomorphism(
         int *A6, int *A4, int verbose_level);
   long int apply_null_polarity(
      long int a, int verbose_level);
   long int apply_polarity(
      long int a, int *Polarity36, int verbose_level);

};


// ##############################################################################
// knarr.cpp
// ##############################################################################

//! the Knarr construction of a GQ from a BLT-set



class knarr {
public:
   int q;
```

```
   int BLT_no;

   W3q *W;
   projective_space *P5;
   grassmann *G63;
   field_theory::finite_field *F;
   long int *BLT;
   int *BLT_line_idx;
   int *Basis;
   int *Basis2;
   int *subspace_basis;
   int *Basis_Pperp;
   ring_theory::longinteger_object *six_choose_three_q;
   int six_choose_three_q_int;
   int f_show;
   int dim_intersection;
   int *Basis_intersection;
   data_structures::fancy_set *type_i_points, *type_ii_points, *type_iii_points;
   data_structures::fancy_set *type_a_lines, *type_b_lines;
   int *type_a_line_BLT_idx;
   int q2;
   int q5;
   int v5[5];
   int v6[6];

   knarr();
   ~knarr();
   void init(
         field_theory::finite_field *F,
         int BLT_no, int verbose_level);
   void points_and_lines(int verbose_level);
   void incidence_matrix(int *&Inc, int &nb_points,
      int &nb_lines, int verbose_level);

};


// ##############################################################################
// object_with_canonical_form.cpp
// ##############################################################################


//! a combinatorial object for which a canonical form can be computed using Nauty
```

```
class object_with_canonical_form {
public:
   projective_space *P;

   object_with_canonical_form_type type;
      // t_PTS = a multiset of points
      // t_LNS = a set of lines
      // t_PNL = a set of points and a set of lines
      // t_PAC = a packing (i.e. q^2+q+1 sets of lines of size q^2+1)
      // t_INC = incidence geometry
      // t_LS = large set

   std::string input_fname;
   int input_idx;
   int f_has_known_ago;
   long int known_ago;

   std::string set_as_string;

   long int *set;
   int sz;
      // set[sz] is used by t_PTS, t_LNS, t_INC

   // for t_PNL:
   long int *set2;
   int sz2;


      // if t_INC or t_LS
   int v;
   int b;
   int f_partition;
   int *partition; // [v + b], do not free !

      // if t_LS
      int design_k;
      int design_sz;

      // t_PAC = packing, uses SoS
      data_structures::set_of_sets *SoS;
      // SoS is used by t_PAC

   data_structures::tally *C;
      // used to determine multiplicities in the set of points

   object_with_canonical_form();
   ~object_with_canonical_form();
```

```
void print(std::ostream &ost);
void print_rows(std::ostream &ost,
      int f_show_incma, int verbose_level);
void print_tex_detailed(std::ostream &ost,
      int f_show_incma, int verbose_level);
void print_tex(std::ostream &ost, int verbose_level);
void get_packing_as_set_system(long int *&Sets,
      int &nb_sets, int &set_size, int verbose_level);
void init_point_set(
      long int *set, int sz,
   int verbose_level);
void init_point_set_from_string(
      std::string &set_text,
      int verbose_level);
void init_line_set(
      long int *set, int sz,
   int verbose_level);
void init_line_set_from_string(
      std::string &set_text,
      int verbose_level);
void init_points_and_lines(
   long int *set, int sz,
   long int *set2, int sz2,
   int verbose_level);
void init_points_and_lines_from_string(
   std::string &set_text,
   std::string &set2_text,
   int verbose_level);
void init_packing_from_set(
   long int *packing, int sz,
   int verbose_level);
void init_packing_from_string(
      std::string &packing_text,
      int q,
      int verbose_level);
void init_packing_from_set_of_sets(
      data_structures::set_of_sets *SoS, int verbose_level);
void init_packing_from_spread_table(
   long int *data,
   long int *Spread_table, int nb_spreads, int spread_size,
   int q,
   int verbose_level);
void init_incidence_geometry(
   long int *data, int data_sz, int v, int b, int nb_flags,
   int verbose_level);
void init_incidence_geometry_from_vector(
   std::vector<int> &Flags, int v, int b, int nb_flags,
```

```
   int verbose_level);
void init_incidence_geometry_from_string(
   std::string &data,
   int v, int b, int nb_flags,
   int verbose_level);
void init_incidence_geometry_from_string_of_row_ranks(
   std::string &data,
   int v, int b, int r,
   int verbose_level);
void init_large_set(
   long int *data, int data_sz, int v, int b, int k, int design_sz,
   int verbose_level);
void init_large_set_from_string(
   std::string &data_text, int v, int k, int design_sz,
   int verbose_level);
void encoding_size(
      int &nb_rows, int &nb_cols,
      int verbose_level);
void encoding_size_point_set(
      int &nb_rows, int &nb_cols,
      int verbose_level);
void encoding_size_line_set(
      int &nb_rows, int &nb_cols,
      int verbose_level);
void encoding_size_points_and_lines(
      int &nb_rows, int &nb_cols,
      int verbose_level);
void encoding_size_packing(
      int &nb_rows, int &nb_cols,
      int verbose_level);
void encoding_size_large_set(
      int &nb_rows, int &nb_cols,
      int verbose_level);
void encoding_size_incidence_geometry(
      int &nb_rows, int &nb_cols,
      int verbose_level);
void canonical_form_given_canonical_labeling(
      int *canonical_labeling,
      data_structures::bitvector *&B,
      int verbose_level);
void encode_incma(
      combinatorics::encoded_combinatorial_object *&Enc,
      int verbose_level);
void encode_point_set(
      combinatorics::encoded_combinatorial_object *&Enc,
      int verbose_level);
void encode_line_set(
```

```
      combinatorics::encoded_combinatorial_object *&Enc,
      int verbose_level);
void encode_points_and_lines(
      combinatorics::encoded_combinatorial_object *&Enc,
      int verbose_level);
void encode_packing(
      combinatorics::encoded_combinatorial_object *&Enc,
      int verbose_level);
void encode_large_set(
      combinatorics::encoded_combinatorial_object *&Enc,
      int verbose_level);
void encode_incidence_geometry(
      combinatorics::encoded_combinatorial_object *&Enc,
      int verbose_level);
void encode_incma_and_make_decomposition(
      combinatorics::encoded_combinatorial_object *&Enc,
      incidence_structure *&Inc,
      data_structures::partitionstack *&Stack,
      int verbose_level);
void encode_object(long int *&encoding, int &encoding_sz,
   int verbose_level);
void encode_object_points(long int *&encoding, int &encoding_sz,
   int verbose_level);
void encode_object_lines(long int *&encoding, int &encoding_sz,
   int verbose_level);
void encode_object_points_and_lines(
      long int *&encoding, int &encoding_sz,
      int verbose_level);
void encode_object_packing(
      long int *&encoding, int &encoding_sz,
   int verbose_level);
void encode_object_incidence_geometry(
      long int *&encoding, int &encoding_sz, int verbose_level);
void encode_object_large_set(
      long int *&encoding, int &encoding_sz, int verbose_level);
void run_nauty(
      int f_compute_canonical_form,
      data_structures::bitvector *&Canonical_form,
      data_structures::nauty_output *&NO,
      int verbose_level);
void canonical_labeling(
      data_structures::nauty_output *NO,
      int verbose_level);
void run_nauty_basic(
      data_structures::nauty_output *&NO,
      int verbose_level);
```

```
};



// ##############################################################################
// point_line.cpp
// ##############################################################################

//! auxiliary class for the class point_line


struct plane_data {
   int *points_on_lines; // [nb_pts * (plane_order + 1)]
   int *line_through_two_points; // [nb_pts * nb_pts]
};


//! a data structure for general projective planes, including non-desarguesian on
es


class point_line {

public:
   data_structures::partitionstack *P;

   int m, n;
   int *a; // the same as in PB
#if 0
   int f_joining;
   int f_point_pair_joining_allocated;
   int m2; // m choose 2
   int *point_pair_to_idx; // [m * m]
   int *idx_to_point_i; // [m choose 2]
   int *idx_to_point_j; // [m choose 2]
   int max_point_pair_joining;
   int *nb_point_pair_joining; // [m choose 2]
   int *point_pair_joining; // [(m choose 2) * max_point_pair_joining]

   int f_block_pair_joining_allocated;
   int n2; // n choose 2
   int *block_pair_to_idx; // [n * n]
   int *idx_to_block_i; // [n choose 2]
   int *idx_to_block_j; // [n choose 2]
   int max_block_pair_joining;
   int *nb_block_pair_joining; // [n choose 2]z
   int *block_pair_joining; // [(n choose 2) * max_block_pair_joining]
```

```
#endif

   // plane_data:
   int f_projective_plane;
   int plane_order; // order = prime ^ exponent
   int plane_prime;
   int plane_exponent;
   int nb_pts;
   int f_plane_data_computed;
      // indicates whether or not plane and dual_plane
      // have been computed by init_plane_data()

   struct plane_data plane;
   struct plane_data dual_plane;

   // data for the coordinatization:
   int line_x_eq_y;
   int line_infty;
   int line_x_eq_0;
   int line_y_eq_0;

   int quad_I, quad_O, quad_X, quad_Y, quad_C;
   int *pt_labels;  // [m]
   int *points;  // [m]
      // pt_labels and points are mutually
      // inverse permutations of {0,1,...,m-1}
      // the affine point (x,y) is labeled as x * plane_order + y

   int *pts_on_line_x_eq_y;  // [plane_order + 1];
   int *pts_on_line_x_eq_y_labels;  // [plane_order + 1];
   int *lines_through_X;  // [plane_order + 1];
   int *lines_through_Y;  // [plane_order + 1];
   int *pts_on_line;  // [plane_order + 1];
   int *MOLS;  // [(plane_order + 1) * plane_order * plane_order]
   int *field_element; // [plane_order]
   int *field_element_inv; // [plane_order]


   int is_desarguesian_plane(int verbose_level);
   int identify_field_not_of_prime_order(int verbose_level);
   void init_projective_plane(int order, int verbose_level);
   void free_projective_plane();
   void plane_report(std::ostream &ost);
   int plane_line_through_two_points(int pt1, int pt2);
   int plane_line_intersection(int line1, int line2);
   void plane_get_points_on_line(int line, int *pts);
   void plane_get_lines_through_point(int pt, int *lines);
```

```
int plane_points_collinear(int pt1, int pt2, int pt3);
int plane_lines_concurrent(int line1, int line2, int line3);
int plane_first_quadrangle(int &pt1, int &pt2, int &pt3, int &pt4);
int plane_next_quadrangle(int &pt1, int &pt2, int &pt3, int &pt4);
int plane_quadrangle_first_i(int *pt, int i);
int plane_quadrangle_next_i(int *pt, int i);
void coordinatize_plane(int O, int I, int X, int Y,
      int *MOLS, int verbose_level);
// needs pt_labels, points, pts_on_line_x_eq_y, pts_on_line_x_eq_y_labels,
// lines_through_X, lines_through_Y, pts_on_line, MOLS to be allocated
int &MOLSsxb(int s, int x, int b);
int &MOLSaddition(int a, int b);
int &MOLSmultiplication(int a, int b);
int ternary_field_is_linear(int *MOLS, int verbose_level);
void print_MOLS(std::ostream &ost);

int is_projective_plane(data_structures::partitionstack &P,
      int &order, int verbose_level);
   // if it is a projective plane, the order is returned.
   // otherwise, 0 is returned.
int count_RC(data_structures::partitionstack &P, int row_cell, int col_cell);
int count_CR(data_structures::partitionstack &P, int col_cell, int row_cell);
int count_RC_representative(
      data_structures::partitionstack &P,
   int row_cell, int row_cell_pt, int col_cell);
int count_CR_representative(
      data_structures::partitionstack &P,
   int col_cell, int col_cell_pt, int row_cell);
int count_pairs_RRC(
      data_structures::partitionstack &P,
      int row_cell1, int row_cell2, int col_cell);
int count_pairs_CCR(
      data_structures::partitionstack &P,
      int col_cell1, int col_cell2, int row_cell);
int count_pairs_RRC_representative(
      data_structures::partitionstack &P,
      int row_cell1, int row_cell_pt, int row_cell2, int col_cell);
   // returns the number of joinings from a point of
   // row_cell1 to elements of row_cell2 within col_cell
   // if that number exists, -1 otherwise
int count_pairs_CCR_representative(
      data_structures::partitionstack &P,
      int col_cell1, int col_cell_pt, int col_cell2, int row_cell);
   // returns the number of joinings from a point of
   // col_cell1 to elements of col_cell2 within row_cell
   // if that number exists, -1 otherwise
void get_MOLm(int *MOLS, int order, int m, int *&M);
```

```
};



// ##############################################################################
// points_and_lines.cpp
// ##############################################################################

//! points and lines in projective space, for instance on a surface



class points_and_lines {

public:

   projective_space *P;

   long int *Pts;
   int nb_pts;


   long int *Lines;
   int nb_lines;

   orthogonal_geometry::quadratic_form *Quadratic_form;
      // if P->n == 3
      // to be able to rank the Pluecker coordinates of lines
      // as points on the Klein quadric




   points_and_lines();
   ~points_and_lines();
   void init(
         projective_space *P,
         std::vector<long int> &Points,
         int verbose_level);
   void unrank_point(int *v, long int rk);
   long int rank_point(int *v);
   void print_all_points(std::ostream &ost);
   void print_all_lines(std::ostream &ost);
   void print_lines_tex(std::ostream &ost);
   void write_points_to_txt_file(std::string &label, int verbose_level);
```

```
};


// ##############################################################################
// polarity.cpp
// ##############################################################################

//! a polarity between points and hyperplanes in PG(n,q)


class polarity {

public:

   projective_space *P;

   int *Point_to_hyperplane; // [P->N_points]
   int *Hyperplane_to_point; // [P->N_points]

   int *f_absolute;  // [P->N_points]

   long int *Line_to_line; // [P->N_lines] only if n = 3
   int *f_absolute_line; // [P->N_lines] only if n = 3
   int nb_absolute_lines;
   int nb_self_dual_lines;


   polarity();
   ~polarity();
   void init_standard_polarity(projective_space *P, int verbose_level);
   void init_general_polarity(projective_space *P, int *Mtx, int verbose_level);
   void determine_absolute_points(int *&f_absolute, int verbose_level);
   void determine_absolute_lines(int verbose_level);
   void init_reversal_polarity(projective_space *P, int verbose_level);
   void report(std::ostream &f);

};



// ##############################################################################
// projective_space_implementation.cpp
// ##############################################################################

//! internal representation of a projective space PG(n,q)
```

```
class projective_space_implementation {

public:

   projective_space *P;

   data_structures::bitmatrix *Bitmatrix;

   int *Lines; // [N_lines * k]
   int *Lines_on_point; // [N_points * r]
   int *Line_through_two_points; // [N_points * N_points]
   int *Line_intersection; // [N_lines * N_lines]

   int *v; // [n + 1]
   int *w; // [n + 1]


   projective_space_implementation();
   ~projective_space_implementation();
   void init(projective_space *P, int verbose_level);
   void line_intersection_type(
         long int *set, int set_size, int *type, int verbose_level);
   void point_types_of_line_set(
         long int *set_of_lines, int set_size,
      int *type, int verbose_level);
   void point_types_of_line_set_int(
      int *set_of_lines, int set_size,
      int *type, int verbose_level);

};


// ##############################################################################
// projective_space_of_dimension_three.cpp
// ##############################################################################

//! projective space PG(3,q) collects functionality specific for a three space


class projective_space_of_dimension_three {

public:
   projective_space *P;

   projective_space_of_dimension_three();
```

```
   ~projective_space_of_dimension_three();
   void init(projective_space *P, int verbose_level);
   void determine_quadric_in_solid(
         long int *nine_pts_or_more, int nb_pts,
      int *ten_coeffs, int verbose_level);
   void quadric_points_brute_force(int *ten_coeffs,
      long int *points, int &nb_points, int verbose_level);
   int point_of_intersection_of_a_line_and_a_line_in_three_space(
      long int line1,
      long int line2, int verbose_level);
   int point_of_intersection_of_a_line_and_a_plane_in_three_space(
      long int line,
      int plane, int verbose_level);
   long int line_of_intersection_of_two_planes_in_three_space(
      long int plane1, long int plane2, int verbose_level);
   long int transversal_to_two_skew_lines_through_a_point(
      long int line1, long int line2, int pt, int verbose_level);
   long int
   line_of_intersection_of_two_planes_in_three_space_using_dual_coordinates(
      long int plane1, long int plane2, int verbose_level);
   void plane_intersection_matrix_in_three_space(
      long int *Planes,
      int nb_planes, int *&Intersection_matrix,
      int verbose_level);
   long int plane_rank_using_dual_coordinates_in_three_space(
      int *eqn4, int verbose_level);
   long int dual_rank_of_plane_in_three_space(long int plane_rank,
      int verbose_level);
   void plane_equation_from_three_lines_in_three_space(
      long int *three_lines,
      int *plane_eqn4, int verbose_level);



};



// ##############################################################################
// projective_space_plane.cpp
// ##############################################################################

//! projective space PG(2,q) collects functionality specific for a desarguesian p
rojective plane


class projective_space_plane {
```

```
public:
   projective_space *P;

   projective_space_plane();
   ~projective_space_plane();
   void init(projective_space *P, int verbose_level);
   int determine_line_in_plane(
         long int *two_input_pts,
      int *three_coeffs,
      int verbose_level);
   int conic_test(
         long int *S, int len, int pt, int verbose_level);
   int test_if_conic_contains_point(
         int *six_coeffs, int pt);
   int determine_conic_in_plane(
         long int *input_pts, int nb_pts,
         int *six_coeffs,
         int verbose_level);
         // returns FALSE if the rank of the
         // coefficient matrix is not 5.
         // TRUE otherwise.
   int determine_cubic_in_plane(
         ring_theory::homogeneous_polynomial_domain *Poly_3_3,
         int nb_pts, long int *Pts, int *coeff10,
         int verbose_level);
   void conic_points_brute_force(int *six_coeffs,
      long int *points, int &nb_points, int verbose_level);
   void conic_points(long int *five_pts, int *six_coeffs,
      long int *points, int &nb_points, int verbose_level);
   void find_tangent_lines_to_conic(int *six_coeffs,
      long int *points, int nb_points,
      long int *tangents, int verbose_level);
   int determine_hermitian_form_in_plane(
         int *pts, int nb_pts,
      int *six_coeffs, int verbose_level);
   void conic_type_randomized(
         int nb_times,
      long int *set, int set_size,
      long int **&Pts_on_conic, int *&nb_pts_on_conic, int &len,
      int verbose_level);
   void conic_intersection_type(
         int f_randomized, int nb_times,
      long int *set, int set_size,
      int threshold,
      int *&intersection_type, int &highest_intersection_number,
      int f_save_largest_sets,
```

```
      data_structures::set_of_sets *&largest_sets,
      int verbose_level);
   void determine_nonconical_six_subsets(
      long int *set, int set_size,
      std::vector<int> &Rk,
      int verbose_level);
   void conic_type(
      long int *set, int set_size,
      int threshold,
      long int **&Pts_on_conic,
      int **&Conic_eqn, int *&nb_pts_on_conic, int &nb_conics,
      int verbose_level);
   void find_nucleus(int *set, int set_size, int &nucleus,
      int verbose_level);
   void points_on_projective_triangle(
         long int *&set, int &set_size,
      long int *three_points, int verbose_level);
   long int dual_rank_of_line_in_plane(
      long int line_rank, int verbose_level);
   long int line_rank_using_dual_coordinates_in_plane(
      int *eqn3, int verbose_level);


};

// ##############################################################################
// projective_space_reporting.cpp
// ##############################################################################

//! internal representation of a projective space PG(n,q)


class projective_space_reporting {

public:

   projective_space *P;

   projective_space_reporting();
   ~projective_space_reporting();
   void init(
         projective_space *P, int verbose_level);
   void create_latex_report(
         graphics::layered_graph_draw_options *O,
         int verbose_level);
   void report_summary(
         std::ostream &ost);
```

```
   void report(
         std::ostream &ost,
         graphics::layered_graph_draw_options *O,
         int verbose_level);
   void report_subspaces_of_dimension(
         std::ostream &ost,
         int vs_dimension, int verbose_level);
   void cheat_sheet_points(
         std::ostream &f, int verbose_level);
   void cheat_polarity(
         std::ostream &f, int verbose_level);
   void cheat_sheet_point_table(
         std::ostream &f, int verbose_level);
   void cheat_sheet_points_on_lines(
         std::ostream &f, int verbose_level);
   void cheat_sheet_lines_on_points(
         std::ostream &f, int verbose_level);
   void cheat_sheet_line_intersection(
         std::ostream &f, int verbose_level);
   void cheat_sheet_line_through_pairs_of_points(
         std::ostream &f,
      int verbose_level);
   void print_set_numerical(
         std::ostream &ost, long int *set, int set_size);
   void print_set(
         long int *set, int set_size);
   void print_line_set_numerical(
         long int *set, int set_size);
   void print_set_of_points(
         std::ostream &ost, long int *Pts, int nb_pts);
   void print_all_points();


};


// ##############################################################################
// projective_space.cpp
// ##############################################################################

//! projective space PG(n,q) of dimension n over Fq


class projective_space {

public:
```

```
grassmann *Grass_lines;
grassmann *Grass_planes; // if n > 2
grassmann *Grass_hyperplanes;
   // if n > 2 (for n=3, planes and
   // hyperplanes are the same thing)

grassmann **Grass_stack; // [n + 1]

field_theory::finite_field *F;
ring_theory::longinteger_object *Go;

int n; // projective dimension
int q;
long int N_points, N_lines;
long int *Nb_subspaces;  // [n + 1]
   // Nb_subspaces[i]
   // = generalized_binomial(n + 1, i + 1, q);
   // N_points = Nb_subspaces[0]
   // N_lines = Nb_subspaces[1];

int r; // number of lines on a point
int k; // number of points on a line

projective_space_implementation *Implementation;

projective_space_plane *Plane; // if n == 2

projective_space_of_dimension_three *Solid; // if n == 3


polarity *Standard_polarity;
polarity *Reversal_polarity;

arc_in_projective_space *Arc_in_projective_space;

projective_space_reporting *Reporting;

int *v; // [n + 1]
int *w; // [n + 1]
int *Mtx; // [3 * (n + 1)], used in is_incident
int *Mtx2; // [3 * (n + 1)], used in is_incident


projective_space();
~projective_space();
void projective_space_init(
      int n,
```

```
      field_theory::finite_field *F,
   int f_init_incidence_structure,
   int verbose_level);
void init_incidence_structure(int verbose_level);
void init_polarity(int verbose_level);
void intersect_with_line(
      long int *set, int set_sz,
      int line_rk,
      long int *intersection, int &sz,
      int verbose_level);
void create_points_on_line(
      long int line_rk, long int *line,
   int verbose_level);
   // needs line[k]
void create_lines_on_point(
   long int point_rk,
   long int *line_pencil, int verbose_level);
void create_lines_on_point_but_inside_a_plane(
   long int point_rk, long int plane_rk,
   long int *line_pencil, int verbose_level);
int create_point_on_line(
   long int line_rk, int pt_rk, int verbose_level);
// pt_rk is between 0 and q-1.
void make_incidence_matrix(int &m, int &n,
   int *&Inc, int verbose_level);
void make_incidence_matrix(
   std::vector<int> &Pts, std::vector<int> &Lines,
   int *&Inc, int verbose_level);
int is_incident(int pt, int line);
void incidence_m_ii(int pt, int line, int a);
void make_incidence_structure_and_partition(
   incidence_structure *&Inc,
   data_structures::partitionstack *&Stack,
   int verbose_level);
void incma_for_type_ij(
   int row_type, int col_type,
   int *&Incma, int &nb_rows, int &nb_cols,
   int verbose_level);
   // row_type, col_type are the vector space
   // dimensions of the objects
   // indexing rows and columns.
int incidence_test_for_objects_of_type_ij(
   int type_i, int type_j, int i, int j,
   int verbose_level);
void points_on_line(long int line_rk,
      long int *&the_points,
      int &nb_points, int verbose_level);
```

```
void points_covered_by_plane(
      long int plane_rk,
      long int *&the_points,
      int &nb_points, int verbose_level);
void incidence_and_stack_for_type_ij(
   int row_type, int col_type,
   incidence_structure *&Inc,
   data_structures::partitionstack *&Stack,
   int verbose_level);
long int nb_rk_k_subspaces_as_lint(int k);
long int rank_point(int *v);
void unrank_point(int *v, long int rk);
void unrank_points(int *v, long int *Rk, int sz);
long int rank_line(int *basis);
void unrank_line(int *basis, long int rk);
void unrank_lines(int *v, long int *Rk, int nb);
long int rank_plane(int *basis);
void unrank_plane(int *basis, long int rk);
long int line_through_two_points(
      long int p1, long int p2);
int test_if_lines_are_disjoint(
      long int l1, long int l2);
int test_if_lines_are_disjoint_from_scratch(
      long int l1, long int l2);
int intersection_of_two_lines(
      long int l1, long int l2);

void line_intersection_type(
      long int *set, int set_size, int *type,
   int verbose_level);
void line_intersection_type_basic(
      long int *set, int set_size, int *type,
   int verbose_level);
   // type[N_lines]
void line_intersection_type_basic_given_a_set_of_lines(
      long int *lines_by_rank, int nb_lines,
   long int *set, int set_size, int *type, int verbose_level);
// type[nb_lines]
void line_intersection_type_through_hyperplane(
   long int *set, int set_size,
   int *type, int verbose_level);
   // type[N_lines]
void point_plane_incidence_matrix(
      long int *point_rks, int nb_points,
      long int *plane_rks, int nb_planes,
      int *&M, int verbose_level);
void plane_intersection_type_basic(
```

```
      long int *set, int set_size,
   int *type, int verbose_level);
   // type[N_planes]
void hyperplane_intersection_type_basic(
      long int *set, int set_size,
   int *type, int verbose_level);
   // type[N_hyperplanes]
void line_intersection_type_collected(
      long int *set, int set_size,
   int *type_collected, int verbose_level);
   // type[set_size + 1]
void find_external_lines(
      long int *set, int set_size,
   long int *external_lines, int &nb_external_lines,
   int verbose_level);
void find_tangent_lines(
      long int *set, int set_size,
   long int *tangent_lines, int &nb_tangent_lines,
   int verbose_level);
void find_secant_lines(
      long int *set, int set_size,
      long int *secant_lines, int &nb_secant_lines,
   int verbose_level);
void find_k_secant_lines(
      long int *set, int set_size, int k,
   long int *secant_lines, int &nb_secant_lines,
   int verbose_level);
void Baer_subline(
      long int *pts3, long int *&pts, int &nb_pts,
   int verbose_level);

int is_contained_in_Baer_subline(long int *pts, int nb_pts,
   int verbose_level);
void export_incidence_matrix_to_csv(int verbose_level);
void make_fname_incidence_matrix_csv(std::string &fname);
void compute_decomposition(
      data_structures::partitionstack *S1,
      data_structures::partitionstack *S2,
      incidence_structure *&Inc,
      data_structures::partitionstack *&Stack,
      int verbose_level);
void compute_decomposition_based_on_tally(
      data_structures::tally *T1,
      data_structures::tally *T2,
      incidence_structure *&Inc,
      data_structures::partitionstack *&Stack,
      int verbose_level);
```

```
void polarity_rank_k_subspace(int k,
      long int rk_in, long int &rk_out, int verbose_level);

// projective_space2.cpp:
void circle_type_of_line_subset(int *pts, int nb_pts,
   int *circle_type, int verbose_level);
   // circle_type[nb_pts]
void intersection_of_subspace_with_point_set(
   grassmann *G, int rk, long int *set, int set_size,
   long int *&intersection_set, int &intersection_set_size,
   int verbose_level);
void intersection_of_subspace_with_point_set_rank_is_longinteger(
   grassmann *G, ring_theory::longinteger_object &rk,
   long int *set, int set_size,
   long int *&intersection_set, int &intersection_set_size,
   int verbose_level);
void plane_intersection_invariant(
      grassmann *G,
   long int *set, int set_size,
   int *&intersection_type, int &highest_intersection_number,
   int *&intersection_matrix, int &nb_planes,
   int verbose_level);
void plane_intersection_type(
   long int *set, int set_size, int threshold,
   intersection_type *&Int_type,
   int verbose_level);
void plane_intersections(
      grassmann *G,
   long int *set, int set_size,
   ring_theory::longinteger_object *&R,
   data_structures::set_of_sets &SoS,
   int verbose_level);
void plane_intersection_type_fast(
      grassmann *G,
   long int *set, int set_size,
   ring_theory::longinteger_object *&R,
   long int **&Pts_on_plane, int *&nb_pts_on_plane, int &len,
   int verbose_level);

void find_planes_which_intersect_in_at_least_s_points(
   long int *set, int set_size,
   int s,
   std::vector<int> &plane_ranks,
   int verbose_level);
void plane_intersection(
      int plane_rank,
      long int *set, int set_size,
```

```
         std::vector<int> &point_indices,
         std::vector<int> &point_local_coordinates,
         int verbose_level);
   void line_intersection(
         int line_rank,
         long int *set, int set_size,
         std::vector<int> &point_indices,
         int verbose_level);
   void line_plane_incidence_matrix_restricted(
         long int *Lines, int nb_lines,
      int *&M, int &nb_planes, int verbose_level);
   int test_if_lines_are_skew(
         int line1, int line2, int verbose_level);
#if 0
   void decomposition_from_set_partition(
         int nb_subsets, int *sz, int **subsets,
      incidence_structure *&Inc,
      data_structures::partitionstack *&Stack,
      int verbose_level);
#endif

   void planes_through_a_line(
      long int line_rk, std::vector<long int> &plane_ranks,
      int verbose_level);



   // projective_space3.cpp:
   int reverse_engineer_semilinear_map(
      int *Elt, int *Mtx, int &frobenius,
      int verbose_level);
   void planes_through_line(long int *Lines, int nb_lines,
         long int *&Plane_ranks,
         int &nb_planes_on_one_line, int verbose_level);


};

// ##############################################################################
// spread_domain.cpp
// ##############################################################################

#define SPREAD_OF_TYPE_FTWKB 1
#define SPREAD_OF_TYPE_KANTOR 2
#define SPREAD_OF_TYPE_KANTOR2 3
#define SPREAD_OF_TYPE_GANLEY 4
#define SPREAD_OF_TYPE_LAW_PENTTILA 5
```

```
#define SPREAD_OF_TYPE_DICKSON_KANTOR 6
#define SPREAD_OF_TYPE_HUDSON 7

//! spreads of PG(k-1,q) in PG(n-1,q) where k divides n


class spread_domain {

public:

   field_theory::finite_field *F;

   int n; // = a multiple of k
   int k;
   int kn; // = k * n
   int q;

   long int nCkq; // = {n choose k}_q
      // used in print_elements, print_elements_and_points
   long int nC1q; // = {n choose 1}_q
   long int kC1q; // = {k choose 1}_q

   long int qn; // q^n
   long int qk; // q^k

   int order; // q^k
   int spread_size; // = order + 1

   long int r;
   long int nb_pts;
   long int nb_points_total; // = nb_pts = {n choose 1}_q
   //long int block_size;
   // = r = {k choose 1}_q, used in spread_lifting.spp

   geometry::grassmann *Grass;
      // {n choose k}_q

   // for check_function and check_function_incremental:
   int *tmp_M1;
   int *tmp_M2;
   int *tmp_M3;
   int *tmp_M4;

   // only if n = 2 * k:
   geometry::klein_correspondence *Klein;
   layer1_foundations::orthogonal_geometry::orthogonal *O;
```

```
int *Data1;
   // for early_test_func
   // [max_depth * kn],
   // previously [Nb * n], which was too much
int *Data2;
   // for early_test_func
   // [n * n]

spread_domain();
~spread_domain();
void init(
      field_theory::finite_field *F,
      int n, int k,
      int verbose_level);
void unrank_point(int *v, long int a);
long int rank_point(int *v);
void unrank_subspace(int *M, long int a);
long int rank_subspace(int *M);
void print_points();
void print_points(long int *pts, int len);
void print_elements();
void print_elements_and_points();
void early_test_func(
      long int *S, int len,
   long int *candidates, int nb_candidates,
   long int *good_candidates, int &nb_good_candidates,
   int verbose_level);
int check_function(
      int len, long int *S, int verbose_level);
int incremental_check_function(
      int len, long int *S, int verbose_level);
void compute_dual_spread(
      int *spread, int *dual_spread,
   int verbose_level);
void print(
      std::ostream &ost, int len, long int *S);
void czerwinski_oakden(
      int level, int verbose_level);
void write_spread_to_file(
      int type_of_spread, int verbose_level);
void make_spread(
      long int *data, int type_of_spread,
      int verbose_level);
void make_spread_from_q_clan(
      long int *data, int type_of_spread,
   int verbose_level);
```

```
   void read_and_print_spread(
         std::string &fname, int verbose_level);
   void HMO(
         std::string &fname, int verbose_level);
   void print_spread(
         std::ostream &ost, long int *data, int sz);

};


// ##############################################################################
// spread_tables.cpp
// ##############################################################################

//! tables with line-spreads in PG(3,q)


class spread_tables {

public:
   int q;
   int d; // = 4
   field_theory::finite_field *F;
   projective_space *P; // PG(3,q)
   grassmann *Gr; // Gr_{4,2}
   long int nb_lines;
   int spread_size;
   int nb_iso_types_of_spreads;

   std::string prefix;

   std::string fname_dual_line_idx;
   std::string fname_self_dual_lines;
   std::string fname_spreads;
   std::string fname_isomorphism_type_of_spreads;
   std::string fname_dual_spread;
   std::string fname_self_dual_spreads;
   std::string fname_schreier_table;

   int *dual_line_idx; // [nb_lines]
   int *self_dual_lines; // [nb_self_dual_lines]
   int nb_self_dual_lines;

   int nb_spreads;
   long int *spread_table; // [nb_spreads * spread_size]
   int *spread_iso_type; // [nb_spreads]
   long int *dual_spread_idx; // [nb_spreads]
```

```
long int *self_dual_spreads; // [nb_self_dual_spreads]
int nb_self_dual_spreads;

int *schreier_table; // [nb_spreads * 4]

spread_tables();
~spread_tables();
void init(projective_space *P,
      int f_load,
      int nb_iso_types_of_spreads,
      std::string &path_to_spread_tables,
      int verbose_level);
void create_file_names(int verbose_level);
void init_spread_table(int nb_spreads,
      long int *spread_table, int *spread_iso_type,
      int verbose_level);
void init_tables(int nb_spreads,
      long int *spread_table, int *spread_iso_type,
      long int *dual_spread_idx,
      long int *self_dual_spreads, int nb_self_dual_spreads,
      int verbose_level);
void init_schreier_table(int *schreier_table,
      int verbose_level);
void init_reduced(
      int nb_select, int *select,
      spread_tables *old_spread_table,
      std::string &path_to_spread_tables,
      int verbose_level);
long int *get_spread(int spread_idx);
void find_spreads_containing_two_lines(std::vector<int> &v,
      int line1, int line2, int verbose_level);

void classify_self_dual_spreads(int *&type,
      data_structures::set_of_sets *&SoS,
      int verbose_level);
int files_exist(int verbose_level);
void save(int verbose_level);
void load(int verbose_level);
void compute_adjacency_matrix(
      data_structures::bitvector *&Bitvec,
      int verbose_level);
int test_if_spreads_are_disjoint(int a, int b);
void compute_dual_spreads(long int **Sets,
      long int *&Dual_spread_idx,
      long int *&self_dual_spread_idx,
      int &nb_self_dual_spreads,
      int verbose_level);
```

```
   int test_if_pair_of_sets_are_adjacent(
         long int *set1, int sz1,
         long int *set2, int sz2,
         int verbose_level);
   int test_if_set_of_spreads_is_line_disjoint(
         long int *set, int len);
   int test_if_set_of_spreads_is_line_disjoint_and_complain_if_not(
         long int *set, int len);
   void make_exact_cover_problem(
         solvers::diophant *&Dio,
         long int *live_point_index, int nb_live_points,
         long int *live_blocks, int nb_live_blocks,
         int nb_needed,
         int verbose_level);
   void compute_list_of_lines_from_packing(
         long int *list_of_lines,
         long int *packing, int sz_of_packing,
         int verbose_level);
   // list_of_lines[sz_of_packing * spread_size]
   void compute_iso_type_invariant(
         int *Partial_packings, int nb_pp, int sz,
         int *&Iso_type_invariant,
         int verbose_level);
   void report_one_spread(std::ostream &ost, int a);

};




// ##############################################################################
// W3q.cpp
// ##############################################################################

//! isomorphism between the W(3,q) and the Q(4,q) generalized quadrangles


class W3q {
public:
   int q;

   projective_space *P3;
   orthogonal_geometry::orthogonal *Q4;
   field_theory::finite_field *F;

   //int *Basis; // [2 * 4]
```

```
   int nb_lines;
      // number of absolute lines of W(3,q)
      // = number of points on Q(4,q)

   int *Lines; // [nb_lines]
      // Lines[] is a list of all absolute lines of PG(3,q)
      // under the chosen symplectic form.
      // The symplectic form is defined
      // in the function evaluate_symplectic_form(),
      // which relies on
      // F->Linear_algebra->evaluate_symplectic_form.
      // The form consists of 2x2 blocks
      // of the form (0,1,-1,0)
      // along the diagonal

   int *Q4_rk; // [nb_lines]
   int *Line_idx; // [nb_lines]
      // Q4_rk[] and Line_idx[] are inverse permutations
      // for a line a, Q4_rk[a] is the point b
      // on the quadric corresponding to it.
      // For a point b on the quadric,
      // Line_idx[b] is the index b of the corresponding line


   W3q();
   ~W3q();
   void init(
         field_theory::finite_field *F, int verbose_level);
   void find_lines(int verbose_level);
   void print_lines();
   int evaluate_symplectic_form(int *x4, int *y4);
   void isomorphism_Q4q(int *x4, int *y4, int *v);
   void print_by_lines();
   void print_by_points();
   int find_line(int line);
};



}}}


#endif /* ORBITER_SRC_LIB_FOUNDATIONS_GEOMETRY_GEOMETRY_H_ */
```

3.11 Geometry Builder

```
/*
 * geometry_builder.h
 *
 *  Created on: Aug 24, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_FOUNDATIONS_GEOMETRY_BUILDER_GEOMETRY_BUILDER_H_
#define SRC_LIB_FOUNDATIONS_GEOMETRY_BUILDER_GEOMETRY_BUILDER_H_


namespace orbiter {
namespace layer1_foundations {
namespace geometry_builder {


#define COLOR_RED 2
#define COLOR_GREEN 3

// pick color codes from the list below:
// the list is taken from void mp_graphics::color_tikz(ofstream &fp, int color)
// line 2600

#if 0
if (color == 0)
   fp << "white";
else if (color == 1)
   fp << "black";
else if (color == 2)
   fp << "red";
else if (color == 3)
   fp << "green";
else if (color == 4)
   fp << "blue";
else if (color == 5)
   fp << "cyan";
else if (color == 6)
   fp << "magenta";
else if (color == 7)
   fp << "pink";
else if (color == 8)
   fp << "orange";
else if (color == 9)
   fp << "lightgray";
else if (color == 10)
   fp << "brown";
```

```
else if (color == 11)
   fp << "lime";
else if (color == 12)
   fp << "olive";
else if (color == 13)
   fp << "gray";
else if (color == 14)
   fp << "purple";
else if (color == 15)
   fp << "teal";
else if (color == 16)
   fp << "violet";
else if (color == 17)
   fp << "darkgray";
else if (color == 18)
   fp << "lightgray";
else if (color == 19)
   fp << "yellow";
else if (color == 20)
   fp << "green!50!red";
else if (color == 21)
   fp << "violet!50!red";
else if (color == 22)
   fp << "cyan!50!red";
else if (color == 23)
   fp << "green!50!blue";
else if (color == 24)
   fp << "brown!50!red";
else if (color == 25)
   fp << "purple!50!red";
else {
#endif
```

```
// ##############################################################################
// cperm.cpp
// ##############################################################################

//! a permutation for use in class gen_geo


class cperm {
```

```
public:
   int l;
   int *data;
      // a permutation of { 0, 1 ... l - 1 }

   cperm();
   ~cperm();
   void init_and_identity(int l);
   void free();
   void move_to(cperm *q);
   void identity();
   void mult(cperm *b, cperm *c);
   void inverse(cperm *b);
   void power(cperm *res, int exp);
   void print();
   void mult_apply_forwc_r(int i, int l);
   /* a := a (i i+1 ... i+l-1). */
   void mult_apply_tau_r(int i, int j);
   /* a := a (i j). */
   void mult_apply_tau_l(int i, int j);
   /* a := (i j) a. */
   void mult_apply_backwc_l(int i, int l);
   /* a := (i+l-1 i+l-2 ... i+1 i) a. */

};
```

```
// ##############################################################################
// decomposition_with_fuse.cpp
// ##############################################################################

//! a row-tactical decomposition with fuse, to be used by the geometry_builder

class decomposition_with_fuse {

public:

   gen_geo *gg;

   int nb_fuse;
   int *Fuse_first; // [nb_fuse]
   int *Fuse_len; // [nb_fuse]
```

```
   int *K0; // [gg->GB->v_len * gg->GB->b_len]
   int *KK; // [gg->GB->v_len * gg->GB->b_len]
   int *K1; // [gg->GB->v_len * gg->GB->b_len]
   int *F_last_k_in_col; // [gg->GB->v_len * gg->GB->b_len]


   gen_geo_conf *Conf; //[gg->GB->v_len * gg->GB->b_len]

   // partition for Nauty:
   int *row_partition;
      // row partition: 1111011110...
      // where the 0´s indicate the end of a block
      // The blocks are defined by the initial TDO decomposition.
   int *col_partition;
      // likewise, but for columns
      // The blocks are defined by the initial TDO decomposition.
   int **Partition;
      // [gg->GB->V + 1]
      // combination of row and column partition,
      // but with only i rows, so that it can be used
      // for computing the canonical form of the partial geometry
      // consisting of the first i rows only
   int **Partition_fixing_last;


   decomposition_with_fuse();
   ~decomposition_with_fuse();
   gen_geo_conf *get_conf_IJ(int I, int J);
   void init(gen_geo *gg, int verbose_level);
   void TDO_init(int *v, int *b, int *theTDO, int verbose_level);
   void init_tdo_line(int fuse_idx,
         int tdo_line, int v, int *b, int *r, int verbose_level);
   void print_conf();
   void init_fuse(int verbose_level);
   void init_k(int verbose_level);
   void conf_init_last_non_zero_flag(int verbose_level);
   void init_partition(int verbose_level);


};



// ##############################################################################
// gen_geo_conf.cpp
// ##############################################################################
```

```
//! description of a configuration which is part of class decomposition_with_fuse



class gen_geo_conf {

public:
   int fuse_idx;

   int v;
   int b;
   int r;

   int r0;
   int i0;
   int j0;
   int f_last_non_zero_in_fuse;
      // only valid if J=0,
      // that is, for those in the first column

   gen_geo_conf();
   ~gen_geo_conf();
   void print(std::ostream &ost);

};

// ##############################################################################
// gen_geo.cpp
// ##############################################################################

//! classification of geometries with a given row-tactical decomposition


class gen_geo {

public:

   geometry_builder *GB;

   decomposition_with_fuse *Decomposition_with_fuse;

   incidence *inc;

   int forget_ivhbar_in_last_isot;

   std::string inc_file_name;
```

```
   // record the search tree in text files for later processing:
   std::string fname_search_tree;
   std::ofstream *ost_search_tree;
   std::string fname_search_tree_flags;
   std::ofstream *ost_search_tree_flags;

   girth_test *Girth_test;

   test_semicanonical *Test_semicanonical;

   geometric_backtrack_search *Geometric_backtrack_search;

   gen_geo();
   ~gen_geo();
   void init(geometry_builder *GB, int verbose_level);
   void init_semicanonical(int verbose_level);
   void print_pairs(int line);
   void main2(int verbose_level);
   void generate_all(int verbose_level);
   void setup_output_files(int verbose_level);
   void close_output_files(int verbose_level);
   void record_tree(int i1, int f_already_there);
   void print_I_m(int I, int m);
   void print(int v);
   void increment_pairs_point(int i1, int col, int k);
   void decrement_pairs_point(int i1, int col, int k);
   void girth_test_add_incidence(int i, int j_idx, int j);
   void girth_test_delete_incidence(int i, int j_idx, int j);
   void girth_Floyd(int i, int verbose_level);
   int check_girth_condition(int i, int j_idx, int j, int verbose_level);
   int apply_tests(int I, int m, int J, int n, int j, int verbose_level);
   void print(std::ostream &ost, int v, int v_cut);
   void print_override_theX(std::ostream &ost,
         int *theX, int v, int v_cut);

};

// ##############################################################################
// geometric_backtrack_search.cpp
// ##############################################################################

//! classification of geometries with a given row-tactical decomposition


class geometric_backtrack_search {
```

```
public:

   gen_geo *gg;

   iso_type **Row_stabilizer_orbits;
   int *Row_stabilizer_orbit_idx;

   geometric_backtrack_search();
   ~geometric_backtrack_search();
   void init(gen_geo *gg, int verbose_level);

   int First(int verbose_level);
   int Next(int verbose_level);
   int BlockFirst(int I, int verbose_level);
   int BlockNext(int I, int verbose_level);
   int RowFirstSplit(int I, int m, int verbose_level);
   int RowNextSplit(int I, int m, int verbose_level);
   int geo_back_test(int I, int verbose_level);
   int RowFirst0(int I, int m, int verbose_level);
   int RowNext0(int I, int m, int verbose_level);
   int RowFirst(int I, int m, int verbose_level);
   int RowNext(int I, int m, int verbose_level);
   int RowFirstLexLeast(int I, int m, int verbose_level);
   int RowNextLexLeast(int I, int m, int verbose_level);
   int RowFirstOrderly(int I, int m, int verbose_level);
   void place_row(int I, int m, int idx, int verbose_level);
   int RowNextOrderly(int I, int m, int verbose_level);
   void RowClear(int I, int m);
   int ConfFirst(int I, int m, int J, int verbose_level);
   int ConfNext(int I, int m, int J, int verbose_level);
   void ConfClear(int I, int m, int J);
   int XFirst(int I, int m, int J, int n, int verbose_level);
   int XNext(int I, int m, int J, int n, int verbose_level);
   void XClear(int I, int m, int J, int n);
   int X_First(int I, int m, int J, int n, int j, int verbose_level);
   int TryToPlace(int I, int m, int J, int n, int j, int verbose_level);

};



// ##############################################################################
// geometry_builder_description.cpp
// ##############################################################################

//! description of a geometry
```

```
class geometry_builder_description {
public:

   int f_V;
   std::string V_text;
   int f_B;
   std::string B_text;
   int f_TDO;
   std::string TDO_text;
   int f_fuse;
   std::string fuse_text;

   int f_girth_test;
   int girth;

   // f_lambda and f_find_square are mutually exclusive!

   int f_lambda;
   int lambda;

   int f_find_square;
   int f_simple;

   int f_search_tree;
   int f_search_tree_flags;

   int f_orderly;
   int f_special_test_not_orderly;

   std::vector<std::string> test_lines;

   std::vector<std::string> test2_lines;

   int f_split;
   int split_line;
   int split_remainder;
   int split_modulo;

   std::vector<int> print_at_line;

   int f_fname_GEO;
   std::string fname_GEO;

   int f_output_to_inc_file;
   int f_output_to_sage_file;
   int f_output_to_blocks_file;
```

```
   int f_output_to_blocks_latex_file;

   geometry_builder_description();
   ~geometry_builder_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// geometry_builder.cpp
// ##############################################################################

//! classification of geometries





class geometry_builder {

public:

   geometry_builder_description *Descr;


   // the row partition:
   int *v;
   int v_len;

   // the column partition:
   int *b;
   int b_len;

   // a coarse grain partition of the row partition
   int *fuse;
   int fuse_len;

   // the structure constants (# of incidences in a row)
   int *TDO;
   int TDO_len;
```

```
   int V;
      // = sum(i = 0; i < v_len; i++) v[i]
   int B;
      // = sum(i= 0; i < b_len; i++) b[i]
   int *V_partition; // [V + 1]

   int *R; // [V]


   int f_transpose_it;
   int f_save_file;
   std::string fname;

   std::string control_file_name;
   int no;
   int flag_numeric;
   int f_no_inc_files;
   gen_geo *gg;

   geometry_builder();
   ~geometry_builder();
   void init_description(
         geometry_builder_description *Descr,
         int verbose_level);
   void compute_VBR(int verbose_level);
   void print_tdo();
   void isot(int line, int verbose_level);
   void isot_no_vhbars(int verbose_level);
   void isot2(int line, int verbose_level);
   void set_split(int line, int remainder, int modulo);

};




// ##############################################################################
// girth_test.cpp
// ##############################################################################

//! classification of geometries
```

```
class girth_test {

public:

   gen_geo *gg;

   int girth;
   int V; // = gg->GB->V

   int **S; // [V][V * V]
   int **D; // [V][V * V]

   girth_test();
   ~girth_test();
   void init(gen_geo *gg, int girth, int verbose_level);
   void Floyd(int row, int verbose_level);
   void add_incidence(int i, int j_idx, int j);
   void delete_incidence(int i, int j_idx, int j);
   int check_girth_condition(
         int i, int j_idx, int j, int verbose_level);
   void print_Si(int i);
   void print_Di(int i);

};




// ##############################################################################
// inc_encoding.cpp
// ##############################################################################

//! row-by-row encoding of an incidence geometry

class inc_encoding {

public:
   int *theX; // [v * dim_n]
   int dim_n;
   int v; // # of rows
   int b; // # of columns
   int *R; // [v]
      // R[i] is the number of incidences in row i

   inc_encoding();
   ~inc_encoding();
   int &theX_ir(int i, int r);
```

```
   void init(int v, int b, int *R, int verbose_level);
   long int rank_row(int row);
   void get_flags(int row, std::vector<int> &flags);
   int find_square(int m, int n);
   void print_horizontal_bar(
      std::ostream &ost,
      gen_geo *gg, int f_print_isot, iso_type *it);
   void print_partitioned(
         std::ostream &ost, int v_cur, int v_cut,
         gen_geo *gg, int f_print_isot);
   void print_partitioned_override_theX(
         std::ostream &ost, int v_cur, int v_cut,
         gen_geo *gg, int *the_X, int f_print_isot);
   void print_permuted(cperm *pv, cperm *qv);
   void apply_permutation(incidence *inc, int v,
      int *theY, cperm *p, cperm *q, int verbose_level);


};




// ##############################################################################
// incidence.cpp
// ##############################################################################

//! encoding of an incidence geometry during classification


class incidence {

public:

   gen_geo *gg;
   inc_encoding *Encoding;

   int *K; //[gg->GB->B]
      // K[j] is the current sum of incidences in column j

   int **theY; //[gg->GB->B][gg->GB->V];

   int **pairs;
      //[gg->GB->V][];
      // pairs[i][i1]
      // is the number of blocks containing {i1,i}
```

```
   // where 0 \le i1 < i.



int gl_nb_GEN;

iso_type **iso_type_at_line; // [gg->GB->V]
iso_type *iso_type_no_vhbars;

int back_to_line;


incidence();
~incidence();
void init(gen_geo *gg,
      int v, int b, int *R, int verbose_level);
void init_pairs(int verbose_level);
void print_pairs(int v);
int find_square(int m, int n);
void print_param();
void free_isot();
void print_R(int v, cperm *p, cperm *q);
void install_isomorphism_test_after_a_given_row(
      int i,
      int f_orderly, int verbose_level);
void install_isomorphism_test_of_second_kind_after_a_given_row(
      int i,
      int f_orderly, int verbose_level);
void set_split(int row, int remainder, int modulo);
void print_geo(
      std::ostream &ost, int v, int *theGEO);
void print_inc(
      std::ostream &ost, int v, long int *theInc);
void print_sage(
      std::ostream &ost, int v, long int *theInc);
void print_blocks(
      std::ostream &ost, int v, long int *theInc);
void compute_blocks(
      long int *&Blocks, int *&K, int v, long int *theInc);
void compute_blocks_ranked(
      long int *&Blocks, int v, long int *theInc);
int compute_k(int v, long int *theInc);
int is_block_tactical(int v, long int *theInc);
void geo_to_inc(
      int v, int *theGEO, long int *theInc, int nb_flags);
void inc_to_geo(
      int v, long int *theInc, int *theGEO, int nb_flags);
```

```
};



// ##############################################################################
// iso_type.cpp
// ##############################################################################

//! classification of geometries based on canonical forms

class iso_type {

public:

   gen_geo *gg;

   int v;
   int sum_R;
   int sum_R_before;

   int f_orderly;

   // test of the first or the second kind:
   // (second kind means we only check those geometries
   // that are completely realizable
   int f_generate_first;
   int f_beginning_checked;

   int f_split;
   int split_remainder;
   int split_modulo;


   std::string fname;

   data_structures::classify_using_canonical_forms *Canonical_forms;

   int f_print_mod;
   int print_mod;

   iso_type();
   ~iso_type();
   void init(gen_geo *gg, int v,
```

```
        int f_orderly, int verbose_level);
   void add_geometry(
      inc_encoding *Encoding,
      int f_partition_fixing_last,
      int &f_already_there,
      int verbose_level);
   void find_and_add_geo(
      int *theY,
      int f_partition_fixing_last,
      int &f_new_object, int verbose_level);
   void second();
   void set_split(int remainder, int modulo);
   void print_geos(int verbose_level);
   void write_inc_file(std::string &fname, int verbose_level);
   void write_sage_file(std::string &fname, int verbose_level);
   void write_blocks_file(std::string &fname, int verbose_level);
   void write_blocks_file_long(std::string &fname, int verbose_level);
   void print_GEO(int *pc, int v, incidence *inc);
   void print_status(std::ostream &ost, int f_with_flags);
   void print_flags(std::ostream &ost);
   void print_geometry(
        inc_encoding *Encoding, int v, incidence *inc);

};




// ##############################################################################
// test_semicanonical.cpp
// ##############################################################################

//! classification of geometries




class test_semicanonical {

public:

   gen_geo *gg;

   int MAX_V;
```

```
   // initial vertical and horizontal bars
   // to create semi-canonical partial geometries
   int nb_i_vbar;
   int *i_vbar;
   int nb_i_hbar;
   int *i_hbar;


   int *f_vbar; // [gg->GB->V * gg->inc->Encoding->dim_n]
   int *vbar; // [gg->GB->V]
   int *hbar; // [gg->GB->B]

   test_semicanonical();
   ~test_semicanonical();
   void init(gen_geo *gg, int MAX_V, int verbose_level);
   void init_bars(int verbose_level);
   void print();
   void markers_update(int I, int m, int J, int n, int j,
         int i1, int j0, int r,
         int verbose_level);
   void marker_move_on(int I, int m, int J, int n, int j,
         int i1, int j0, int r,
         int verbose_level);
   int row_starter(int I, int m, int J, int n,
         int i1, int j0, int r,
         int verbose_level);
   void row_init(int I, int m, int J,
         int i1,
         int verbose_level);
   int col_marker_test(int j0, int j, int i1);
   void col_marker_remove(int I, int m, int J, int n,
         int i1, int j0, int r, int old_x);
   void row_test_continue(int I, int m, int J, int i1);

};







}}}


#endif /* SRC_LIB_FOUNDATIONS_GEOMETRY_BUILDER_GEOMETRY_BUILDER_H_ */
```

3.12 Graph Theory

```
// graph_theory.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005


#ifndef ORBITER_SRC_LIB_FOUNDATIONS_GRAPH_THEORY_GRAPH_THEORY_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_GRAPH_THEORY_GRAPH_THEORY_H_



namespace orbiter {
namespace layer1_foundations {
namespace graph_theory {



// ##############################################################################
// clique_finder_control.cpp
// ##############################################################################


#define CLIQUE_FINDER_CONTROL_MAX_RESTRICTIONS 100

//! settings that control the clique finding process


class clique_finder_control {

public:
   int f_rainbow;

   int f_target_size;
   int target_size;

   int f_weighted;
   int weights_total;
   int weights_offset;
   std::string weights_string;
   std::string weights_bounds;
```

```
int f_Sajeeb;
int f_nonrecursive;
int f_output_solution_raw;
int f_store_solutions;

int f_output_file;
std::string output_file;

int f_maxdepth;
int maxdepth;

int f_restrictions;
int nb_restrictions;
int restrictions[CLIQUE_FINDER_CONTROL_MAX_RESTRICTIONS * 3];

int f_tree;
int f_decision_nodes_only;
std::string fname_tree;

int print_interval;

// extra stuff for the clique finder
// that does not come from the command line:

int f_has_additional_test_function;
void (*call_back_additional_test_function)(
   rainbow_cliques *R, void *user_data,
   int current_clique_size, int *current_clique,
   int nb_pts, int &reduced_nb_pts,
   int *pt_list, int *pt_list_inv,
   int verbose_level);
void *additional_test_function_data; // previously user_data

int f_has_print_current_choice_function;
void (*call_back_print_current_choice)(clique_finder *CF,
   int depth, void *user_data, int verbose_level);
void *print_current_choice_data; // previously user_data



// output variables:
unsigned long int nb_search_steps;
unsigned long int nb_decision_steps;
int dt;

int *Sol;
long int nb_sol;
```

```
   clique_finder_control();
   ~clique_finder_control();
   int parse_arguments(
         int argc, std::string *argv);
   void print();

};




// ##############################################################################
// clique_finder.cpp
// ##############################################################################

//! finds all cliques of a certain size in a graph



class clique_finder {
public:


   clique_finder_control *Control;


   std::string label;
   int n; // number of points


   int f_write_tree;
   std::string fname_tree;
   std::ofstream *fp_tree;


   int *point_labels;
   int *point_is_suspicious;

   int verbose_level;


   int f_has_adj_list;
   int *adj_list_coded;
   int f_has_bitvector;
```

```
data_structures::bitvector *Bitvec_adjacency;

int f_has_row_by_row_adjacency_matrix;
char **row_by_row_adjacency_matrix; // [n][n]


int *pt_list;
int *pt_list_inv;
int *nb_points;
int *candidates; // [max_depth * n]
int *nb_candidates; // [max_depth]
int *current_choice; // [max_depth]
int *level_counter; // [max_depth] (added Nov 8, 2014)

// restrictions for partial search
int *f_level_mod; // [max_depth] (added Nov 8, 2014)
int *level_r; // [max_depth] (added Nov 8, 2014)
int *level_m; // [max_depth] (added Nov 8, 2014)

int *current_clique; // [max_depth]

unsigned long int counter; // number of backtrack nodes
unsigned long int decision_step_counter;
   // number of backtrack nodes that are decision nodes

// solution storage:
std::deque<std::vector<int> > solutions;
long int nb_sol;


// callbacks:
void (*call_back_clique_found)(clique_finder *CF, int verbose_level);

// added May 26, 2009:
void (*call_back_add_point)(clique_finder *CF,
   int current_clique_size, int *current_clique,
   int pt, int verbose_level);
void (*call_back_delete_point)(clique_finder *CF,
   int current_clique_size, int *current_clique,
   int pt, int verbose_level);
int (*call_back_find_candidates)(clique_finder *CF,
   int current_clique_size, int *current_clique,
   int nb_pts, int &reduced_nb_pts,
   int *pt_list, int *pt_list_inv,
   int *candidates, int verbose_level);
   // Jan 2, 2012: added reduced_nb_pts and pt_list_inv
```

```
int (*call_back_is_adjacent)(clique_finder *CF,
   int pt1, int pt2, int verbose_level);

// added Oct 2011:
void (*call_back_after_reduction)(clique_finder *CF,
   int depth, int nb_points, int verbose_level);


void *call_back_clique_found_data1;
void *call_back_clique_found_data2;


clique_finder();
~clique_finder();
void null();
void free();
void init(clique_finder_control *Control,
      std::string &label, int n,
      int f_has_adj_list, int *adj_list_coded,
      int f_has_bitvector,
      data_structures::bitvector *Bitvec_adjacency,
      int verbose_level);
void init_restrictions(int *restrictions, int verbose_level);
void init_point_labels(int *pt_labels);
void init_suspicious_points(int nb, int *point_list);
void backtrack_search(int depth, int verbose_level);
int solve_decision_problem(int depth, int verbose_level);
   // returns TRUE if we found a solution
//void backtrack_search_not_recursive(int verbose_level);
void open_tree_file(std::string &fname_base);
void close_tree_file();
void get_solutions(
      int *&Sol, long int &nb_solutions, int &clique_sz,
   int verbose_level);
void print_suspicious_points();
void print_set(int size, int *set);
void print_suspicious_point_subset(int size, int *set);
void log_position_and_choice(int depth,
      unsigned long int  counter_save,
      unsigned long int counter);
void log_position(int depth,
      unsigned long int  counter_save,
      unsigned long int counter);
void log_choice(int depth);
void swap_point(int idx1, int idx2);
void degree_of_point_statistic(int depth, int nb_points,
   int verbose_level);
```

```
   int degree_of_point(int depth, int i, int nb_points);
   int is_suspicious(int i);
   int point_label(int i);
   int is_adjacent(int depth, int i, int j);
   int is_viable(int depth, int pt);
   void write_entry_to_tree_file(int depth, int verbose_level);
   int s_ij(int i, int j);
   void delinearize_adjacency_list(int verbose_level);

private:
   void parallel_delinearize_adjacency_list();
};





// ##############################################################################
// colored_graph.cpp
// ##############################################################################


//! a graph with a vertex coloring


class colored_graph {
public:


   std::string fname_base;

   std::string label;
   std::string label_tex;

   int nb_points;
   int nb_colors;
   int nb_colors_per_vertex; // = 1 by default

   long int L;

   long int *points; // [nb_points]
   int *point_color; // [nb_points * nb_colors_per_vertex]


   int user_data_size;
   long int *user_data; // [user_data_size]
```

```
int f_ownership_of_bitvec;
data_structures::bitvector *Bitvec;

int f_has_list_of_edges;
int nb_edges;
int *list_of_edges;
   // used in early_test_func_for_path_and_cycle_search

colored_graph();
~colored_graph();
void compute_edges(int verbose_level);
int is_adjacent(int i, int j);
void set_adjacency(int i, int j, int a);
void set_adjacency_k(long int k, int a);
void partition_by_color_classes(
   int *&partition, int *&partition_first,
   int &partition_length,
   int verbose_level);
colored_graph *sort_by_color_classes(int verbose_level);
colored_graph *subgraph_by_color_classes(
      int c, int verbose_level);
colored_graph *subgraph_by_color_classes_with_condition(
      int *seed_pts, int nb_seed_pts,
      int c, int verbose_level);
void print();
void print_points_and_colors();
void print_adjacency_list();
void init(
      int nb_points, int nb_colors, int nb_colors_per_vertex,
   int *colors, data_structures::bitvector *Bitvec,
   int f_ownership_of_bitvec,
   std::string &label, std::string &label_tex,
   int verbose_level);
void init_with_point_labels(
      int nb_points, int nb_colors, int nb_colors_per_vertex,
   int *colors, data_structures::bitvector *Bitvec,
   int f_ownership_of_bitvec,
   long int *point_labels,
   std::string &label, std::string &label_tex,
   int verbose_level);
void init_no_colors(
      int nb_points, data_structures::bitvector *Bitvec,
   int f_ownership_of_bitvec,
   std::string &label, std::string &label_tex,
   int verbose_level);
void init_adjacency(
```

```
      int nb_points, int nb_colors, int nb_colors_per_vertex,
   int *colors, int *Adj,
   std::string &label, std::string &label_tex,
   int verbose_level);
void init_adjacency_upper_triangle(
      int nb_points, int nb_colors, int nb_colors_per_vertex,
   int *colors, int *Adj,
   std::string &label, std::string &label_tex,
   int verbose_level);
void init_adjacency_no_colors(int nb_points, int *Adj,
      std::string &label, std::string &label_tex,
   int verbose_level);
void init_adjacency_two_colors(int nb_points,
   int *Adj, int *subset, int sz,
   std::string &label, std::string &label_tex,
   int verbose_level);
void init_user_data(
      long int *data, int data_size, int verbose_level);
void save(std::string &fname, int verbose_level);
void load(std::string &fname, int verbose_level);
void draw_on_circle(
      std::string &fname,
      graphics::layered_graph_draw_options *Draw_options,
   int verbose_level);
void draw_on_circle_2(
      graphics::mp_graphics &G,
   graphics::layered_graph_draw_options *Draw_options);
void create_bitmatrix(data_structures::bitmatrix *&Bitmatrix,
   int verbose_level);
void draw(
      std::string &fname,
      graphics::layered_graph_draw_options *Draw_options,
   int verbose_level);
void draw_Levi(
      std::string &fname,
      graphics::layered_graph_draw_options *Draw_options,
   int f_partition, int nb_row_parts, int *row_part_first,
   int nb_col_parts, int *col_part_first,
   int m, int n, int f_draw_labels,
   int verbose_level);
void draw_with_a_given_partition(
      std::string &fname,
      graphics::layered_graph_draw_options *Draw_options,
      int *parts, int nb_parts,
      int verbose_level);
void draw_partitioned(
      std::string &fname,
```

```
      graphics::layered_graph_draw_options *Draw_options,
   int f_labels,
   int verbose_level);
colored_graph *compute_neighborhood_subgraph(
   int pt,
   data_structures::fancy_set *&vertex_subset,
   data_structures::fancy_set *&color_subset,
   int verbose_level);
colored_graph *compute_neighborhood_subgraph_based_on_subset(
   long int *subset, int subset_sz,
   data_structures::fancy_set *&vertex_subset,
   data_structures::fancy_set *&color_subset,
   int verbose_level);
void export_to_magma(std::string &fname, int verbose_level);
void export_to_maple(std::string &fname, int verbose_level);
void export_to_file(std::string &fname, int verbose_level);
void export_to_text(std::string &fname, int verbose_level);
void export_laplacian_to_file(
      std::string &fname,
   int verbose_level);
void export_to_file_matlab(
      std::string &fname, int verbose_level);
void export_to_csv(
      std::string &fname, int verbose_level);
void export_to_graphviz(
      std::string &fname, int verbose_level);
void early_test_func_for_clique_search(
      long int *S, int len,
   long int *candidates, int nb_candidates,
   long int *good_candidates, int &nb_good_candidates,
   int verbose_level);
void early_test_func_for_coclique_search(
      long int *S, int len,
   long int *candidates, int nb_candidates,
   long int *good_candidates, int &nb_good_candidates,
   int verbose_level);
void early_test_func_for_path_and_cycle_search(
      long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
   int verbose_level);
int is_cycle(int nb_e, long int *edges, int verbose_level);
void all_cliques(
      clique_finder_control *Control,
      std::string &graph_label, int verbose_level);
void all_cliques_rainbow(
      clique_finder_control *Control,
```

```
        std::ostream &ost_txt,
        std::ostream &ost_csv,
        int verbose_level);
   void find_subgraph(
        std::string &subgraph_label, int verbose_level);
   void find_subgraph_E6(int verbose_level);
   void all_cliques_black_and_white(
        clique_finder_control *Control,
        std::ostream &ost_txt,
        std::ostream &ost_csv,
        int verbose_level);
   void write_solutions_to_csv_file(
        clique_finder_control *Control,
        std::ostream &ost, int verbose_level);
   void do_Sajeeb(
        clique_finder_control *Control,
        int verbose_level);
   void do_Sajeeb_black_and_white(
        clique_finder_control *Control,
        std::vector<std::vector<long int> >& solutions,
        int verbose_level);
   void all_cliques_weighted_with_two_colors(
        clique_finder_control *Control,
        int verbose_level);
   void all_cliques_of_size_k_ignore_colors(
        clique_finder_control *Control,
        int verbose_level);
   void all_rainbow_cliques(
        clique_finder_control *Control,
        std::ostream &fp,
        int verbose_level);
   void complement(int verbose_level);
   void distance_2(int verbose_level);
   void properties(int verbose_level);
   int test_distinguishing_property(long int *set, int sz,
        int verbose_level);
   void eigenvalues(double *&E, int verbose_level);
   void Laplace_eigenvalues(double *&E, int verbose_level);

};

void call_back_clique_found_using_file_output(clique_finder *CF,
   int verbose_level);



// ##############################################################################
```

```
// graph_layer.cpp
// ##############################################################################


//! part of the data structure layered_graph



class graph_layer {
public:
   int id_of_first_node;
   int nb_nodes;
   graph_node *Nodes;
   double y_coordinate;

   graph_layer();
   ~graph_layer();
   void init(int nb_nodes,
         int id_of_first_node, int verbose_level);
   void place(int verbose_level);
   void place_with_grouping(
         int *group_size, int nb_groups,
      double x_stretch, int verbose_level);
   void scale_x_coordinates(
         double x_stretch, int verbose_level);
   void write_memory_object(
         orbiter_kernel_system::memory_object *m,
         int verbose_level);
   void read_memory_object(
         orbiter_kernel_system::memory_object *m,
         int verbose_level);
};

// ##############################################################################
// graph_node.cpp
// ##############################################################################

//! part of the data structure layered_graph


class graph_node {
public:
   std::string label;
   int id;

   int f_has_data1;
   int data1;
```

```
int f_has_data2;
int data2;

int f_has_data3;
int data3;

int f_has_vec_data;
long int *vec_data;
int vec_data_len;

int f_has_distinguished_element; // refers to vec_data
int distinguished_element_index;

int layer;
int neighbor_list_allocated;
int nb_neighbors;
int *neighbor_list;
double x_coordinate;

// added June 28, 2016:
int nb_children;
int nb_children_allocated;
int *child_id; // [nb_children]
int weight_of_subtree;
double width;
int depth_first_node_rank;

// added May 25, 2017
double radius_factor;

graph_node();
~graph_node();
void add_neighbor(int l, int n, int id);
void add_text(std::string &text);
void add_vec_data(long int *v, int len);
void set_distinguished_element(int idx);
void add_data1(int data);
void add_data2(int data);
void add_data3(int data);
void write_memory_object(
      orbiter_kernel_system::memory_object *m,
      int verbose_level);
void read_memory_object(
      orbiter_kernel_system::memory_object *m,
      int verbose_level);
void allocate_tree_structure(int verbose_level);
```

```
   int remove_neighbor(
         layered_graph *G, int id, int verbose_level);
   void find_all_parents(
         layered_graph *G, std::vector<int> &All_Parents,
         int verbose_level);
   int find_parent(layered_graph *G, int verbose_level);
   void register_child(
         layered_graph *G, int id_child,
         int verbose_level);
   void place_x_based_on_tree(
         layered_graph *G,
         double left, double right,
      int verbose_level);
   void depth_first_rank_recursion(
         layered_graph *G, int &r,
      int verbose_level);
   void scale_x_coordinate(
         double x_stretch, int verbose_level);

};


// ##############################################################################
// graph_theory_domain.cpp
// ##############################################################################


//! various functions related to graph theory


class graph_theory_domain {
public:
   graph_theory_domain();
   ~graph_theory_domain();

   void colored_graph_draw(
         std::string &fname,
         graphics::layered_graph_draw_options *Draw_options,
         int f_labels,
         int verbose_level);
   void colored_graph_all_cliques(
         clique_finder_control *Control,
         std::string &fname,
         int f_output_solution_raw,
         int f_output_fname, std::string &output_fname,
         int verbose_level);
   void colored_graph_all_cliques_list_of_cases(
```

```
      clique_finder_control *Control,
      long int *list_of_cases, int nb_cases,
      std::string &fname_template,
      std::string &fname_sol,
      std::string &fname_stats,
      int f_split, int split_r, int split_m,
      int f_prefix, std::string &prefix,
      int verbose_level);
void save_as_colored_graph_easy(
      std::string &fname_base,
      int n, int *Adj, int verbose_level);
void save_colored_graph(
      std::string &fname,
      int nb_vertices, int nb_colors,
      int nb_colors_per_vertex,
      long int *points, int *point_color,
      long int *data, int data_sz,
      data_structures::bitvector *Bitvec,
      int verbose_level);
void load_colored_graph(
      std::string &fname,
      int &nb_vertices, int &nb_colors,
      int &nb_colors_per_vertex,
      long int *&vertex_labels,
      int *&vertex_colors, long int *&user_data,
      int &user_data_size,
      data_structures::bitvector *&Bitvec,
      int verbose_level);
int is_association_scheme(
      int *color_graph, int n, int *&Pijk,
   int *&colors, int &nb_colors,
   int verbose_level);
void print_Pijk(int *Pijk, int nb_colors);
void compute_decomposition_of_graph_wrt_partition(
      int *Adj, int N,
   int *first, int *len, int nb_parts, int *&R,
   int verbose_level);
void draw_bitmatrix(
      std::string &fname_base,
      graphics::layered_graph_draw_options *Draw_options,
      int f_dots,
      int f_partition, int nb_row_parts, int *row_part_first,
      int nb_col_parts, int *col_part_first,
      int f_row_grid, int f_col_grid,
      int f_bitmatrix, data_structures::bitmatrix *Bitmatrix,
      int *M, int m, int n,
      int f_has_labels, int *labels,
```

```
      int verbose_level);
void list_parameters_of_SRG(int v_max, int verbose_level);
void make_cycle_graph(
      int *&Adj, int &N,
      int n, int verbose_level);
void make_inversion_graph(
      int *&Adj, int &N,
      int *perm, int n, int verbose_level);
void make_Hamming_graph(
      int *&Adj, int &N,
      int n, int q, int verbose_level);
void make_Johnson_graph(
      int *&Adj, int &N,
      int n, int k, int s, int verbose_level);
void make_Paley_graph(
      int *&Adj, int &N,
      field_theory::finite_field *Fq, int verbose_level);
void make_Schlaefli_graph(
      int *&Adj, int &N,
      field_theory::finite_field *F,
      int verbose_level);
void make_Winnie_Li_graph(int *&Adj, int &N,
      field_theory::finite_field *Fq,
      int index, int verbose_level);
void make_Grassmann_graph(
      int *&Adj, int &N,
      int n, int k,
      field_theory::finite_field *F,
      int r, int verbose_level);
void make_orthogonal_collinearity_graph(
      int *&Adj, int &N,
      int epsilon, int d,
      field_theory::finite_field *F, int verbose_level);
void make_non_attacking_queens_graph(
      int *&Adj, int &N,
      int n, int verbose_level);
void make_disjoint_sets_graph(
      int *&Adj, int &N,
      std::string &fname, int verbose_level);
void compute_adjacency_matrix(
      int *Table, int nb_sets, int set_size,
      std::string &prefix_for_graph,
      data_structures::bitvector *&B,
      int verbose_level);
void make_graph_of_disjoint_sets_from_rows_of_matrix(
   int *M, int m, int n,
   int *&Adj, int verbose_level);
```

```
   void all_cliques_of_given_size(int *Adj,
         int nb_pts, int clique_sz, int *&Sol, long int &nb_sol,
         int verbose_level);
   void eigenvalues(
         graph_theory::colored_graph *CG, int verbose_level);

};




// ##############################################################################
// layered_graph.cpp
// ##############################################################################


//! a data structure to store layered graphs or Hasse diagrams


class layered_graph {
public:
   int nb_layers;
   int nb_nodes_total;
   int id_of_first_node;
   graph_layer *L;
   std::string fname_base;
   int f_has_data1;
   int data1;

   layered_graph();
   ~layered_graph();
   void init(int nb_layers, int *Nb_nodes_layer,
         std::string &fname_base, int verbose_level);
   int nb_nodes();
   void print_nb_nodes_per_level();
   double average_word_length();
   void place(int verbose_level);
   void place_with_y_stretch(
         double y_stretch, int verbose_level);
   void scale_x_coordinates(
         double x_stretch, int verbose_level);
   void place_with_grouping(
         int **Group_sizes, int *Nb_groups,
      double x_stretch,
      int verbose_level);
   void add_edge(
         int l1, int n1, int l2, int n2,
```

```
      int verbose_level);
void add_text(
      int l, int n, std::string &text,
      int verbose_level);
void add_data1(
      int data, int verbose_level);
void add_node_vec_data(
      int l, int n, long int *v, int len,
   int verbose_level);
void set_distinguished_element_index(int l, int n,
   int index, int verbose_level);
void add_node_data1(
      int l, int n, int data,
      int verbose_level);
void add_node_data2(
      int l, int n, int data,
      int verbose_level);
void add_node_data3(
      int l, int n, int data,
      int verbose_level);
void draw_with_options(std::string &fname,
      graphics::layered_graph_draw_options *O,
      int verbose_level);
void coordinates_direct(
      double x_in, double y_in,
   int x_max, int y_max, int f_rotated,
   int &x, int &y);
void coordinates(
      int id, int x_max, int y_max,
   int f_rotated,
   int &x, int &y);
void find_node_by_id(int id, int &l, int &n);
void write_file(
      std::string &fname, int verbose_level);
void read_file(
      std::string &fname, int verbose_level);
void write_memory_object(
      orbiter_kernel_system::memory_object *m,
      int verbose_level);
void read_memory_object(
      orbiter_kernel_system::memory_object *m,
      int verbose_level);
void remove_edges(int layer1, int node1,
      int layer2, int node2,
      std::vector<std::vector<int> > &All_Paths,
      int verbose_level);
void remove_edge(int layer1, int node1,
```

```
         int layer2, int node2,
         int verbose_level);
   void find_all_paths_between(int layer1, int node1,
         int layer2, int node2,
         std::vector<std::vector<int> > &All_Paths,
         int verbose_level);
   void find_all_paths_between_recursion(
         int layer1, int node1,
         int layer2, int node2,
         int l0, int n0,
         std::vector<std::vector<int> > &All_Paths,
         std::vector<int> &Path,
         int verbose_level);
   void create_spanning_tree(
         int f_place_x, int verbose_level);
   void compute_depth_first_ranks(int verbose_level);
   void set_radius_factor_for_all_nodes_at_level(int lvl,
      double radius_factor, int verbose_level);
   void make_subset_lattice(
         int n, int depth, int f_tree,
      int f_depth_first, int f_breadth_first,
      int verbose_level);
   void init_poset_from_file(
         std::string &fname,
         int f_grouping, double x_stretch,
         int verbose_level);
};


// ##############################################################################
// rainbow_cliques.cpp
// ##############################################################################



//! to search for rainbow cliques in graphs




class rainbow_cliques {
public:

   clique_finder_control *Control;

   std::ostream *ost_sol;
```

```
   colored_graph *graph;
   clique_finder *CF;
   int *f_color_satisfied;
   int *color_chosen_at_depth;
   int *color_frequency;

   rainbow_cliques();
   ~rainbow_cliques();

   void search(clique_finder_control *Control,
         colored_graph *graph,
         std::ostream &ost_sol,
         int verbose_level);
   int find_candidates(
      int current_clique_size, int *current_clique,
      int nb_pts, int &reduced_nb_pts,
      int *pt_list, int *pt_list_inv,
      int *candidates, int verbose_level);
   void clique_found(int *current_clique,
         int verbose_level);
   void clique_found_record_in_original_labels(
         int *current_clique,
      int verbose_level);

};

void call_back_colored_graph_clique_found(clique_finder *CF,
   int verbose_level);
void call_back_colored_graph_add_point(clique_finder *CF,
   int current_clique_size, int *current_clique,
   int pt, int verbose_level);
void call_back_colored_graph_delete_point(clique_finder *CF,
   int current_clique_size, int *current_clique,
   int pt, int verbose_level);
int call_back_colored_graph_find_candidates(clique_finder *CF,
   int current_clique_size, int *current_clique,
   int nb_pts, int &reduced_nb_pts,
   int *pt_list, int *pt_list_inv,
   int *candidates, int verbose_level);


}}}


#endif /* ORBITER_SRC_LIB_FOUNDATIONS_GRAPH_THEORY_GRAPH_THEORY_H_ */
```

3.13 Graph Theory Nauty Interface

```
// graph_theory_nauty.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005




#ifndef ORBITER_SRC_LIB_FOUNDATIONS_GRAPH_THEORY_NAUTY_GRAPH_THEORY_NAUTY_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_GRAPH_THEORY_NAUTY_GRAPH_THEORY_NAUTY_H_



namespace orbiter {
namespace layer1_foundations {



// ##############################################################################
// nauty_interface.cpp
// ##############################################################################

//! low-level interface to the graph canonization software nauty

class nauty_interface {

public:

   void nauty_interface_graph_bitvec(int v,
         data_structures::bitvector *Bitvec,
      int *partition,
      data_structures::nauty_output *NO,
      int verbose_level);
   void nauty_interface_graph_int(int v, int *Adj,
      int *partition,
      data_structures::nauty_output *NO,
      int verbose_level);
   void nauty_interface_matrix_int(
      combinatorics::encoded_combinatorial_object *Enc,
      data_structures::nauty_output *NO,
      int verbose_level);
```

```
};

}}

#endif /* ORBITER_SRC_LIB_FOUNDATIONS_GRAPH_THEORY_NAUTY_GRAPH_THEORY_NAUTY_H_ */
```

3.14 Graphics

```
// graphics.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005


#ifndef ORBITER_SRC_LIB_FOUNDATIONS_GRAPHICS_GRAPHICS_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_GRAPHICS_GRAPHICS_H_



namespace orbiter {
namespace layer1_foundations {
namespace graphics {


// ##############################################################################
// animate.cpp
// ##############################################################################


//! creates 3D animations using Povray


class animate {
public:
   povray_job_description *Povray_job_description;
   scene *S;
   std::string output_mask;
   char fname_makefile[1000];
   int nb_frames;
   video_draw_options *Opt;
   std::ofstream *fpm;
   void (*draw_frame_callback)(animate *A, int frame,
               int nb_frames_this_round, int round,
               double clipping,
               std::ostream &fp,
               int verbose_level);
   void *extra_data;
   povray_interface *Pov;

   animate();
```

```
~animate();
void init(
      povray_job_description *Povray_job_description,
      void *extra_data,
      int verbose_level);
void animate_one_round(
   int round,
   int verbose_level);
void draw_single_line(int line_idx, std::string &color,
      std::ostream &fp);
void draw_single_quadric(int idx, std::string &color,
      std::ostream &fp);
void draw_single_surface(int surface_idx, std::ostream &fp);
void draw_single_surface_with_color(
      int surface_idx, std::string &color, std::ostream &fp);
void draw_Hilbert_point(
      int point_idx, double rad,
      std::string &options, std::ostream &fp);
void draw_Hilbert_line(int line_idx, std::string &color,
      std::ostream &fp);
void draw_Hilbert_plane(int plane_idx, std::string &color,
      std::ostream &fp);
void draw_Hilbert_red_line(int idx_one_based,
      std::ostream &fp);
void draw_Hilbert_blue_line(int idx_one_based,
      std::ostream &fp);
void draw_Hilbert_red_lines(std::ostream &fp);
void draw_Hilbert_blue_lines(std::ostream &fp);
void draw_Hilbert_cube_extended_edges(std::ostream &fp);
void draw_Hilbert_cube_faces(std::ostream &fp);
void draw_Hilbert_cube_boxed(std::ostream &fp);
void draw_Hilbert_tetrahedron_boxed(std::ostream &fp);
void draw_Hilbert_tetrahedron_faces(std::ostream &fp);
void draw_frame_Hilbert(
   int h, int nb_frames, int round,
   double clipping_radius,
   std::ostream &fp,
   int verbose_level);
void draw_surface_13_1(std::ostream &fp);
void draw_frame_Hilbert_round_76(
      video_draw_options *Opt,
      int h, int nb_frames, int round,
      std::ostream &fp,
      int verbose_level);
   // tritangent plane, 6 arc points, 2 blue lines, 6 red lines, text
void draw_frame_Eckardt_surface(
   int h, int nb_frames, int round,
```

```
   double clipping_radius,
   std::ostream &fp,
   int verbose_level);
void draw_frame_E4_surface(
   int h, int nb_frames, int round,
   double clipping_radius,
   std::ostream &fp,
   int verbose_level);
void draw_frame_triangulation_of_cube(
   int h, int nb_frames, int round,
   double clipping_radius,
   std::ostream &fp,
   int verbose_level);
void draw_frame_twisted_cubic(
   int h, int nb_frames, int round,
   double clipping_radius,
   std::ostream &fp,
   int verbose_level);
void draw_frame_five_plus_one(
   int h, int nb_frames, int round,
   double clipping_radius,
   std::ostream &fp,
   int verbose_level);
void draw_frame_windy(
   int h, int nb_frames, int round,
   double clipping_radius,
   std::ostream &fp,
   int verbose_level);
void rotation(
      int h, int nb_frames, int round,
      std::ostream &fp);
void union_end(
      int h, int nb_frames, int round,
      double clipping_radius,
      std::ostream &fp);
void draw_text(std::string &text,
      double thickness_half, double extra_spacing,
      double scale,
      double off_x, double off_y, double off_z,
      std::string &color_options,
      int idx_point,
      std::ostream &ost, int verbose_level);
void draw_text_with_selection(int *selection, int nb_select,
   double thickness_half, double extra_spacing,
   double scale,
   double off_x, double off_y, double off_z,
   std::string &options, std::string &group_options,
```

```
      std::ostream &ost, int verbose_level);
};


// ##############################################################################
// draw_bitmap_control.cpp
// ##############################################################################


//! options for drawing bitmap files


class draw_bitmap_control {

public:

   int f_input_csv_file;
   std::string input_csv_file_name;

   int f_secondary_input_csv_file;
   std::string secondary_input_csv_file_name;

   int f_input_object;
   std::string input_object_label;

   int f_input_matrix;
   int *M;
   int *M2;

   int m;
   int n;
   int f_partition;
   int part_width;
   std::string part_row;
   std::string part_col;

   int f_box_width;
   int box_width;

   int f_invert_colors;
   int bit_depth;

   int f_grayscale;


   draw_bitmap_control();
   ~draw_bitmap_control();
```

```
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();
};


// ##############################################################################
// draw_incidence_structure_description.cpp
// ##############################################################################


//! options for drawing an incidence structure


class draw_incidence_structure_description {
public:
   int f_width;
   int width;

   int f_width_10;
   int width_10;

   // width for one box in 0.1mm
   // width_10 is 1 10th of width
   // example: width = 40, width_10 = 4 */

   int f_outline_thin;

   int f_unit_length;
   std::string unit_length;

   int f_thick_lines;
   std::string thick_lines;

   int f_thin_lines;
   std::string thin_lines;

   int f_geo_line_width;
   std::string geo_line_width;

   int v;
   int b;
   int V;
   int B;
   int *Vi;
   int *Bj;
```

```
   int f_labelling_points;
   std::string *point_labels;

   int f_labelling_blocks;
   std::string *block_labels;

// width for one box in 0.1mm
// width_10 is 1 10th of width
// example: width = 40, width_10 = 4

   draw_incidence_structure_description();
   ~draw_incidence_structure_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};

// ##############################################################################
// draw_mod_n_description.cpp
// ##############################################################################


//! options for drawing modulo n


class draw_mod_n_description {
public:

   int f_n;
   int n;

   int f_mod_s;
   int mod_s;

   int f_divide_out_by;
   int divide_out_by;

   int f_file;
   std::string fname;
   int f_inverse;
   int f_additive_inverse;
   int f_power_cycle;
   int power_cycle_base;
```

```
   int f_cyclotomic_sets;
   int cyclotomic_sets_q;
   std::string cyclotomic_sets_reps;

   int f_cyclotomic_sets_thickness;
   int cyclotomic_sets_thickness;

   int f_eigenvalues;
   double eigenvalues_A[4];


   draw_mod_n_description();
   ~draw_mod_n_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// draw_projective_curve_description.cpp
// ##############################################################################


//! options for drawing a projective curve


class draw_projective_curve_description {
public:

   int f_number;
   int number;

   int f_file;
   std::string fname;

   int f_animate;
   int animate_nb_of_steps;

   int f_animate_with_transition;
   int animate_transition_nb_of_steps;

   int f_title_page;
   int f_trailer_page;
```

```
   draw_projective_curve_description();
   ~draw_projective_curve_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// drawable_set_of_objects.cpp
// ##############################################################################



//! a set of objects that should be drawn with certain povray properties



class drawable_set_of_objects {

public:

   int group_idx;

   int type;
   // 1 = sphere
   // 2 = cylinder
   // 3 = prisms (faces)
   // 4 = planes
   // 5 = lines
   // 6 = cubics
   // 7 = quadrics
   // 8 = quartics
   // 9 = quintics
   // 10 = octics
   // 11 = label


   double d;
   double d2; // for text: scale

   std::string properties;
```

```
   drawable_set_of_objects();
   ~drawable_set_of_objects();
   void init_spheres(int group_idx, double rad,
         std::string &properties, int verbose_level);
   void init_cylinders(int group_idx,
         double rad, std::string &properties, int verbose_level);
   void init_prisms(int group_idx,
         double thickness,
         std::string &properties, int verbose_level);
   void init_planes(int group_idx,
         std::string &properties, int verbose_level);
   void init_lines(int group_idx,
         double rad, std::string &properties, int verbose_level);
   void init_cubics(int group_idx,
         std::string &properties, int verbose_level);
   void init_quadrics(int group_idx,
         std::string &properties, int verbose_level);
   void init_quartics(int group_idx,
         std::string &properties, int verbose_level);
   void init_quintics(int group_idx,
         std::string &properties, int verbose_level);
   void init_octics(int group_idx,
         std::string &properties, int verbose_level);
   void init_labels(int group_idx,
         double thickness_half, double scale,
         std::string &properties, int verbose_level);
   void draw(animate *Anim, std::ostream &ost,
         int f_group_is_animated, int frame, int verbose_level);

};


// ##############################################################################
// graphical_output.cpp
// ##############################################################################

//! a catch-all class for things related to 2D graphics


class graphical_output {

private:

public:

   polish::function_polish *smooth_curve_Polish;
```

```
double parabola_a;
double parabola_b;
double parabola_c;


graphical_output();
~graphical_output();
void draw_layered_graph_from_file(std::string &fname,
      layered_graph_draw_options *Opt,
      int verbose_level);
void do_domino_portrait(int D, int s,
      std::string &photo_label,
      layered_graph_draw_options *Opt,
      int verbose_level);
void do_create_points_on_quartic(
      double desired_distance, int verbose_level);
void do_create_points_on_parabola(
      double desired_distance, int N,
      double a, double b, double c, int verbose_level);
void do_smooth_curve(std::string &curve_label,
      double desired_distance, int N,
      double t_min, double t_max, double boundary,
      polish::function_polish_description *FP_descr,
      int verbose_level);
void draw_bitmap(draw_bitmap_control *C, int verbose_level);
void random_noise_in_bitmap_file(
      std::string fname_input,
      std::string fname_output,
      int probability_numerator,
      int probability_denominator,
      int verbose_level);
void random_noise_in_bitmap_file_burst(
      std::string fname_input,
      std::string fname_output,
      int probability_numerator,
      int probability_denominator,
      int burst_length_max,
      int verbose_level);
void draw_projective_curve(
      draw_projective_curve_description *Descr,
      layered_graph_draw_options *Opt, int verbose_level);
void draw_projective(mp_graphics &G,
      int number, int animate_step, int animate_nb_of_steps,
   int f_transition, int transition_step, int transition_nb_steps,
   int f_title_page, int title_page_step,
   int f_trailer_page, int trailer_page_step);
void tree_draw(
```

```
        tree_draw_options *Tree_draw_options, int verbose_level);
   void animate_povray(
        povray_job_description *Povray_job_description,
        int verbose_level);

};


// ##############################################################################
// layered_graph_draw_options.cpp
// ##############################################################################

//! options for drawing an object of type layered_graph

class layered_graph_draw_options {
public:

   int f_paperheight;
   int paperheight;
   int f_paperwidth;
   int paperwidth;

   int xin;
   int yin;
   int xout;
   int yout;


   int f_spanning_tree;


   int f_circle;
   int f_corners;
   int rad;
   int f_embedded;
   int f_sideways;
   int f_show_level_info;
   int f_label_edges;
   int f_x_stretch;
   double x_stretch;
   int f_y_stretch;
   double y_stretch;

   int f_scale;
   double scale;

   int f_line_width;
```

```
   double line_width;

   int f_rotated;


   int f_nodes;
   int f_nodes_empty;


   int f_select_layers;
   std::string select_layers;
   int nb_layer_select;
   int *layer_select;


   int f_has_draw_begining_callback;
   void (*draw_begining_callback)(
         graph_theory::layered_graph *LG, mp_graphics *G,
      int x_max, int y_max, int f_rotated, int dx, int dy);
   int f_has_draw_ending_callback;
   void (*draw_ending_callback)(
         graph_theory::layered_graph *LG, mp_graphics *G,
      int x_max, int y_max, int f_rotated, int dx, int dy);
   int f_has_draw_vertex_callback;
   void (*draw_vertex_callback)(
         graph_theory::layered_graph *LG, mp_graphics *G,
      int layer, int node, int x, int y, int dx, int dy);

   int f_paths_in_between;
   int layer1, node1;
   int layer2, node2;

   layered_graph_draw_options();
   ~layered_graph_draw_options();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();
};


// ##############################################################################
// mp_graphics.cpp
// ##############################################################################

//! a class to help with drawing elements in a 2D grid fashion
```

```
struct grid_frame {
   int f_matrix_notation;
   double origin_x;
   double origin_y;
   int m; // number of rows in the grid
   int n; // number of columns in the grid
   double dx;
   double dy;
};


//! a general 2D graphical output interface (metapost, tikz, postscript)


class mp_graphics {

   layered_graph_draw_options *Draw_options;

   //std::string fname_base;
   std::string fname_mp;
   std::string fname_log;
   std::string fname_tikz;
   std::ofstream fp_mp;
   std::ofstream fp_log;
   std::ofstream fp_tikz;
   int f_file_open;


   // coordinate systems:

   int user[4]; // llx/lly/urx/ury
   int dev[4]; // llx/lly/urx/ury

   int x_min, x_max, y_min, y_max, f_min_max_set;

   int txt_halign;
      // 0=left aligned,
      // 1=centered,
      // 2=right aligned;
      // default=0
   int txt_valign; // 0=bottom, 1=middle, 2=top; default=0
   int txt_boxed; //  default=0
   int txt_overwrite; // default=0
   int txt_rotate; // default = 0 (in degree)

   int line_beg_style; // default=0
   int line_end_style; // 0=nothing, 1=arrow; default=0
```

```
   int line_thickness; // 1,2,3
   int line_color; // 0=white, 1=black, 2=red, 3=green
      // see mp_graphics::color_tikz for tikz colors

   int fill_interior;
      // in 1/100th,
      // 0= none (used for pie-drawing);
      // default=0
   int fill_color; // 0 = white, 1 = black; default=0
   int fill_shape; // 0 =  .., 1 = -- ; default=1
   int fill_outline; // default=0
   int fill_nofill; // default=0


   int line_dashing;
      // 0 = no dashing,
      // otherwise scaling factor 1/100th evenly
      // default=0

   int cur_path;


public:

   mp_graphics();
   ~mp_graphics();
   void init(
         std::string &file_name,
         layered_graph_draw_options *Draw_options,
      int verbose_level);
   void exit(std::ostream &ost, int verbose_level);
   void frame(double move_out);
   void frame_constant_aspect_ratio(double move_out);
   void finish(std::ostream &ost, int verbose_level);

   int& in_xmin();
   int& in_ymin();
   int& in_xmax();
   int& in_ymax();
   int& out_xmin();
   int& out_ymin();
   int& out_xmax();
   int& out_ymax();

   void user2dev(int &x, int &y);
   void dev2user(int &x, int &y);
```

```
void user2dev_dist_x(int &x);
void user2dev_dist_y(int &y);

void draw_polar_grid(double r_max, int nb_circles,
   int nb_rays, double x_stretch);
void draw_axes_and_grid(
      layered_graph_draw_options *O,
   double x_min, double x_max,
   double y_min, double y_max,
   double dx, double dy,
   int f_x_axis_at_y_min, int f_y_axis_at_x_min,
   int x_mod, int y_mod, int x_tick_mod, int y_tick_mod,
   double x_labels_offset, double y_labels_offset,
   double x_tick_half_width, double y_tick_half_width,
   int f_v_lines, int subdivide_v,
   int f_h_lines, int subdivide_h,
   int verbose_level);
void plot_curve(int N, int *f_DNE,
   double *Dx, double *Dy, double dx, double dy);
void nice_circle(int x, int y, int rad);
void grid_polygon2(grid_frame *F, int x0, int y0,
   int x1, int y1);
void grid_polygon4(grid_frame *F, int x0, int y0,
   int x1, int y1, int x2, int y2, int x3, int y3);
void grid_polygon5(grid_frame *F, int x0, int y0,
   int x1, int y1, int x2, int y2,
   int x3, int y3, int x4, int y4);
void polygon(int *Px, int *Py, int n);
void polygon2(int *Px, int *Py, int i1, int i2);
void polygon3(int *Px, int *Py, int i1, int i2, int i3);
void polygon4(int *Px, int *Py, int i1, int i2, int i3,
   int i4);
void polygon5(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5);
void polygon6(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6);
void polygon7(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7);
void polygon8(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7, int i8);
void polygon9(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7, int i8, int i9);
void polygon10(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7, int i8, int i9,
   int i10);
void polygon11(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7, int i8, int i9,
```

```
   int i10, int i11);
void polygon_idx(int *Px, int *Py, int *Idx, int n);
void bezier(int *Px, int *Py, int n);
void bezier2(int *Px, int *Py, int i1, int i2);
void bezier3(int *Px, int *Py, int i1, int i2, int i3);
void bezier4(int *Px, int *Py, int i1, int i2, int i3,
   int i4);
void bezier5(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5);
void bezier6(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6);
void bezier7(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7);
void bezier_idx(int *Px, int *Py, int *Idx, int n);
void grid_fill_polygon4(grid_frame *F,
   int x0, int y0, int x1, int y1, int x2,
   int y2, int x3, int y3);
void grid_fill_polygon5(grid_frame *F,
   int x0, int y0, int x1, int y1,
   int x2, int y2, int x3, int y3,
   int x4, int y4);
void fill_polygon3(int *Px, int *Py, int i1, int i2, int i3);
void fill_polygon4(int *Px, int *Py, int i1, int i2, int i3,
   int i4);
void fill_polygon5(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5);
void fill_polygon6(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6);
void fill_polygon7(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7);
void fill_polygon8(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7, int i8);
void fill_polygon9(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7, int i8, int i9);
void fill_polygon10(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7, int i8, int i9, int i10);
void fill_polygon11(int *Px, int *Py, int i1, int i2, int i3,
   int i4, int i5, int i6, int i7, int i8,
   int i9, int i10, int i11);
void polygon2_arrow_halfway(
      int *Px, int *Py, int i1, int i2);
void polygon2_arrow_halfway_and_label(
      int *Px, int *Py, int i1, int i2,
   const char *alignment, std::string &s);
void grid_aligned_text(grid_frame *F, int x, int y,
   const char *alignment, std::string &s);
void aligned_text(
```

```
      int x, int y, const char *alignment, std::string &s);
void aligned_text_array(int *Px, int *Py, int idx,
   const char *alignment, std::string &s);
void aligned_text_with_offset(
      int x, int y, int xoffset, int yoffset,
   const char *alignment, std::string &s);

void st_alignment(int txt_halign, int txt_valign);
void sl_udsty(int line_dashing);
void sl_ends(int line_beg_style, int line_end_style);
void sl_thickness(int line_thickness);
void sl_color(int line_color);
void sf_interior(int fill_interior);
void sf_color(int fill_color);
void sf_shape(int fill_shape);
void sf_outline(int fill_outline);
void sf_nofill(int fill_nofill);
void st_boxed(int txt_boxed);
void st_overwrite(int txt_overwrite);
void st_rotate(int txt_rotate);
void coords_min_max(int x, int y);

// output commands:
void header();
void footer();
void begin_figure(int factor_1000);
void end_figure();

void comment(std::string &s);
void text(int x, int y, std::string &s);
void circle(int x, int y, int rad);
void circle_text(int x, int y, int rad, std::string &s);
void polygon_idx2(int *Px, int *Py, int *Idx, int n,
      int f_cycle);
void bezier_idx2(int *Px, int *Py, int *Idx, int n,
      int f_cycle);
void fill_idx(int *Px, int *Py, int *Idx, int n,
   const char *symbol, int f_cycle);


// output commands log file:
void header_log(std::string &str_date);
void footer_log();
void comment_log(std::string &s);
void st_alignment_log();
void sl_udsty_log();
void sl_ends_log();
```

```
void sl_thickness_log();
void sl_color_log();
void sf_interior_log();
void sf_color_log();
void sf_shape_log();
void sf_outline_log();
void sf_nofill_log();
void st_boxed_log();
void st_overwrite_log();
void st_rotate_log();
void bezier_idx_log(int *Px, int *Py, int *Idx, int n);
void polygon_log(int *Px, int *Py, int n);
void polygon_idx_log(int *Px, int *Py, int *Idx, int n);
void text_log(int x1, int y1, std::string &s);
void circle_log(int x1, int y1, int rad);


// output commands metapost:
void header_mp(std::string &str_date);
void footer_mp();
void comment_mp(std::string &s);
void text_mp(int x1, int y1, std::string &s);
void begin_figure_mp(int factor_1000);
void end_figure_mp();
void circle_mp(int x, int y, int rad);
void output_circle_text_mp(int x, int y, int idx, std::string &s);
void polygon_idx_mp(int *Px, int *Py,
      int *Idx, int n, int f_cycle);
void bezier_idx_mp(int *Px, int *Py,
      int *Idx, int n, int f_cycle);
void color_tikz(std::ofstream &fp, int color);
void fill_idx_mp(int *Px, int *Py, int *Idx, int n,
   const char *symbol, int f_cycle);
void output_xy_metapost(int x, int y);
void output_x_metapost(int x);
void output_y_metapost(int y);
int get_label(int x, int y);
void get_alignment_mp(char *align);
void line_thickness_mp();

// output commands tikz:
void header_tikz(std::string &str_date);
void footer_tikz();
void comment_tikz(std::string &s);
void text_tikz(int x1, int y1, std::string &s);
void circle_tikz(int x, int y, int rad);
void output_circle_text_tikz(int x, int y, int idx, int rad,
```

```
   const char *text);
void polygon_idx_tikz(int *Px, int *Py,
      int *Idx, int n, int f_cycle);
void bezier_idx_tikz(int *Px, int *Py,
      int *Idx, int n, int f_cycle);
void fill_idx_tikz(std::ofstream &fp,
   int *Px, int *Py, int *Idx, int n,
   const char *symbol, int f_cycle);
void output_xy_tikz(int x, int y);
void output_x_tikz(int x);
void output_y_tikz(int y);

void polygon3D(int *Px, int *Py,
      int dim, int x0, int y0, int z0, int x1, int y1, int z1);
void integer_4pts(int *Px, int *Py,
      int p1, int p2, int p3, int p4,
      const char *align, int a);
void text_4pts(int *Px, int *Py, int p1, int p2, int p3, int p4,
   const char *align, std::string &s);


void draw_graph(int x, int y,
      int dx, int dy, int nb_V,
      long int *Edges, int nb_E, int radius,
      int verbose_level);
void draw_graph_with_distinguished_edge(
   int x, int y,
   int dx, int dy, int nb_V, long int *Edges, int nb_E,
   int distinguished_edge, int verbose_level);
void draw_graph_on_multiple_circles(int x, int y,
      int dx, int dy, int nb_V,
      int *Edges, int nb_E, int nb_circles);
void draw_graph_on_2D_grid(
      int x, int y, int dx, int dy, int rad, int nb_V,
      int *Edges, int nb_E, int *coords_2D, int *Base,
      int f_point_labels,
      int point_label_offset, int f_directed);
void draw_tournament(int x, int y,
      int dx, int dy, int nb_V, long int *Edges, int nb_E,
      int radius,
      int verbose_level);
void draw_bitmatrix2(int f_dots,
   int f_partition,
   int nb_row_parts, int *row_part_first,
   int nb_col_parts, int *col_part_first,
   int f_row_grid, int f_col_grid,
   int f_bitmatrix,
```

```
   data_structures::bitmatrix *Bitmatrix, int *M,
   int m, int n,
   int f_has_labels, int *labels);

void draw_density2(int no,
   int *outline_value, int *outline_number, int outline_sz,
   int min_value, int max_value,
   int offset_x, int f_switch_x,
   int f_title, std::string &title,
   std::string &label_x,
   int f_circle, int circle_at, int circle_rad,
   int f_mu, int f_sigma, int nb_standard_deviations,
   int f_v_grid, int v_grid, int f_h_grid, int h_grid);
void draw_density2_multiple_curves(int no,
   int **outline_value, int **outline_number,
   int *outline_sz, int nb_curves,
   int min_x, int max_x, int min_y, int max_y,
   int offset_x, int f_switch_x,
   int f_title, std::string &title,
   std::string &label_x,
   int f_v_grid, int v_grid, int f_h_grid, int h_grid,
   int f_v_logarithmic, double log_base);
void projective_plane_draw_grid2(
      layered_graph_draw_options *O,
      int q,
      int *Table, int nb,
      int f_point_labels,
      std::string *Point_labels, int verbose_level);
void draw_matrix_in_color(
   int f_row_grid, int f_col_grid,
   int *Table, int nb_colors,
   int m, int n,
   int *color_scale, int nb_colors_in_scale,
   int f_has_labels, int *labels);
void domino_draw1(int M,
      int i, int j, int dx, int dy, int rad, int f_horizontal);
void domino_draw2(int M,
      int i, int j, int dx, int dy, int rad, int f_horizontal);
void domino_draw3(int M,
      int i, int j, int dx, int dy, int rad, int f_horizontal);
void domino_draw4(int M,
      int i, int j, int dx, int dy, int rad, int f_horizontal);
void domino_draw5(int M,
      int i, int j, int dx, int dy, int rad, int f_horizontal);
void domino_draw6(int M,
      int i, int j, int dx, int dy, int rad, int f_horizontal);
void domino_draw7(int M,
```

```
        int i, int j, int dx, int dy, int rad, int f_horizontal);
   void domino_draw8(int M,
        int i, int j, int dx, int dy, int rad, int f_horizontal);
   void domino_draw9(int M,
        int i, int j, int dx, int dy, int rad, int f_horizontal);
   void domino_draw_assignment_East(int Ap, int Aq, int M,
        int i, int j, int dx, int dy, int rad);
   void domino_draw_assignment_South(int Ap, int Aq, int M,
        int i, int j, int dx, int dy, int rad);
   void domino_draw_assignment(int *A, int *matching, int *B,
        int M, int N,
        int dx, int dy,
        int rad, int edge,
        int f_grid, int f_gray, int f_numbers, int f_frame,
        int f_cost, int cost);
};


// ##############################################################################
// parametric_curve_point.cpp
// ##############################################################################

//! an individual point on a continuous curve, sampled through parametric_curve

class parametric_curve_point {
public:

   double t;
   int f_is_valid;
   std::vector<double> coords;

   parametric_curve_point();
   ~parametric_curve_point();
   void init(double t, int f_is_valid, double *x,
        int nb_dimensions, int verbose_level);
};

// ##############################################################################
// parametric_curve.cpp
// ##############################################################################

//! a continuous curve sampled by individual points

class parametric_curve {
public:

   int nb_dimensions;
```

```
   double desired_distance;
   double t0, t1; // parameter interval
   int (*compute_point_function)(double t,
         double *pt, void *extra_data, int verbose_level);
   void *extra_data;
   double boundary;

   int nb_pts;
   std::vector<parametric_curve_point> Pts;

   parametric_curve();
   ~parametric_curve();
   void init(int nb_dimensions,
         double desired_distance,
         double t0, double t1,
         int (*compute_point_function)(double t,
               double *pt, void *extra_data, int verbose_level),
         void *extra_data,
         double boundary,
         int N,
         int verbose_level);

};

// ##############################################################################
// plot_tools.cpp
// ##############################################################################

//! utility functions for plotting (graphing)


class plot_tools {

public:
   plot_tools();
   ~plot_tools();

   void draw_density(
         layered_graph_draw_options *Draw_options,
         std::string &prefix, int *the_set, int set_size,
      int f_title, std::string &title, int out_of,
      std::string &label_x,
      int f_circle, int circle_at, int circle_rad,
      int f_mu, int f_sigma, int nb_standard_deviations,
      int f_v_grid, int v_grid, int f_h_grid, int h_grid,
      int offset_x,
      int f_switch_x, int no, int f_embedded,
```

```
      int verbose_level);
   void draw_density_multiple_curves(
         layered_graph_draw_options *Draw_options,
         std::string &prefix,
      int **Data, int *Data_size, int nb_data_sets,
      int f_title, std::string &title, int out_of,
      std::string &label_x,
      int f_v_grid, int v_grid, int f_h_grid, int h_grid,
      int offset_x, int f_switch_x,
      int f_v_logarithmic, double log_base, int no, int f_embedded,
      int verbose_level);
   void get_coord(int *Px, int *Py, int idx, int x, int y,
      int min_x, int min_y, int max_x, int max_y, int f_switch_x);
   void get_coord_log(int *Px, int *Py, int idx, int x, int y,
      int min_x, int min_y, int max_x, int max_y,
      double log_base, int f_switch_x);
   void y_to_pt_on_curve(int y_in, int &x, int &y,
      int *outline_value, int *outline_number, int outline_sz);
   void projective_plane_draw_grid(std::string &fname,
         layered_graph_draw_options *O,
         int q, int *Table, int nb,
         int f_point_labels, std::string *Point_labels,
         int verbose_level);
   void draw_mod_n(draw_mod_n_description *Descr,
         layered_graph_draw_options *O,
         int verbose_level);
   void draw_mod_n_work(mp_graphics &G,
         layered_graph_draw_options *O,
         draw_mod_n_description *Descr,
         int verbose_level);
   void draw_point_set_in_plane(
      std::string &fname,
      layered_graph_draw_options *O,
      geometry::projective_space *P,
      long int *Pts, int nb_pts,
      int f_point_labels,
      int verbose_level);

};



// ##############################################################################
// povray_interface.cpp
// ##############################################################################

//! povray interface for 3D graphics
```

```
class povray_interface {
public:


   std::string color_white_simple;
   std::string color_white;
   std::string color_white_very_transparent;
   std::string color_black;
   std::string color_pink;
   std::string color_pink_transparent;
   std::string color_green;
   std::string color_gold;
   std::string color_red;
   std::string color_blue;
   std::string color_yellow;
   std::string color_yellow_transparent;
   std::string color_scarlet;
   std::string color_brown;
   std::string color_orange;
   std::string color_orange_transparent;
   std::string color_orange_no_phong;
   std::string color_chrome;
   std::string color_gold_dode;
   std::string color_gold_transparent;
   std::string color_red_wine_transparent;
   std::string color_yellow_lemon_transparent;

   double sky[3];
   double location[3];
   double look_at[3];


   povray_interface();
   ~povray_interface();
   void beginning(std::ostream &ost,
         double angle,
         double *sky,
         double *location,
         double *look_at,
         int f_with_background);
   void animation_rotate_around_origin_and_1_1_1(std::ostream &ost);
   void animation_rotate_around_origin_and_given_vector(double *v,
         std::ostream &ost);
   void animation_rotate_xyz(
```

```
      double angle_x_deg, double angle_y_deg, double angle_z_deg,
      std::ostream &ost);
   void animation_rotate_around_origin_and_given_vector_by_a_given_angle(
      double *v, double angle_zero_one, std::ostream &ost);
   void union_start(std::ostream &ost);
   void union_end(std::ostream &ost,
         double scale_factor, double clipping_radius);
   void union_end_box_clipping(std::ostream &ost, double scale_factor,
         double box_x, double box_y, double box_z);
   void union_end_no_clipping(std::ostream &ost, double scale_factor);
   void bottom_plane(std::ostream &ost);
   void rotate_111(int h, int nb_frames, std::ostream &fp);
   void rotate_around_z_axis(
         int h, int nb_frames, std::ostream &fp);
   void ini(std::ostream &ost,
         const char *fname_pov, int first_frame,
      int last_frame);
};

// ##############################################################################
// povray_job_description.cpp
// ##############################################################################

//! description of a povray job



class povray_job_description {
public:

   int f_output_mask;
   std::string output_mask;
   int f_nb_frames_default;
   int nb_frames_default;
   int f_round;
   int round;
   int f_rounds;
   std::string rounds_as_string;
   video_draw_options *Video_draw_options;

   // for povray_worker:
   scene *S;

   povray_job_description();
   ~povray_job_description();
   int read_arguments(
      int argc, std::string *argv,
```

```
      int verbose_level);
   void print();

};


// ##############################################################################
// scene_element_of_type_edge.cpp
// ##############################################################################


//! a scene element of type edge



class scene_element_of_type_edge {

private:

   std::vector<std::string> Idx;
      // labels of two points

public:
   scene_element_of_type_edge();
   ~scene_element_of_type_edge();
   void init(std::string &pt1, std::string &pt2);
   void print();

};





// ##############################################################################
// scene_element_of_type_face.cpp
// ##############################################################################


//! a scene element of type face



class scene_element_of_type_face {

private:
```

```
   std::vector<std::string> Pts;
      // labels of the points

public:
   scene_element_of_type_face();
   ~scene_element_of_type_face();
   void init(std::vector<std::string> &pts);
   void print();

};


// ##############################################################################
// scene_element_of_type_line.cpp
// ##############################################################################


//! a scene element of type line


class scene_element_of_type_line {

private:

   double Line_coords[6];
   // a line is given by two points

public:
   scene_element_of_type_line();
   ~scene_element_of_type_line();
   void init(double *coord6);
   void print();

};


// ##############################################################################
// scene_element_of_type_plane.cpp
// ##############################################################################
```

```
//! a scene element of type plane



class scene_element_of_type_plane {

private:

   double Plane_coords[4];

public:
   scene_element_of_type_plane();
   ~scene_element_of_type_plane();
   void init(double *coord4);
   void print();

};



// ##############################################################################
// scene_element_of_type_point.cpp
// ##############################################################################


//! a scene element of type point



class scene_element_of_type_point {

private:

   double Point_coords[3];

public:
   scene_element_of_type_point();
   ~scene_element_of_type_point();
   void init(double *coord3);
   void print();

};
```

```
// ##############################################################################
// scene_element_of_type_surface.cpp
// ##############################################################################


//! a scene element of type surface



class scene_element_of_type_surface {

private:

   int d;
   int nb_coeffs;
   double *Eqn;

public:
   scene_element_of_type_surface();
   ~scene_element_of_type_surface();
   void init(int d, int nb_coeffs, double *coords);
   void print();

};





// ##############################################################################
// scene.cpp
// ##############################################################################

#define SCENE_MAX_LINES    100000
#define SCENE_MAX_EDGES    100000
#define SCENE_MAX_POINTS   200000
#define SCENE_MAX_PLANES    10000
#define SCENE_MAX_FACES    200000


#define SCENE_MAX_QUADRICS  10000
#define SCENE_MAX_OCTICS      100
#define SCENE_MAX_QUARTICS   1000
#define SCENE_MAX_QUINTICS    500
```

```
#define SCENE_MAX_CUBICS    10000


//! a collection of 3D geometry objects



class scene {

private:

   double *Line_coords;
      // [nb_lines * 6] a line is given by two points

   int *Edge_points;
      // [nb_edges * 2]

   double *Point_coords;
      // [nb_points * 3]

   double *Plane_coords;
      // [nb_planes * 4]
      // the four parameters A,B,C,D as needed for the povray command
      // plane{<A,B,C>, D}

   double *Quadric_coords;
      // [nb_quadrics * 10]

   double *Cubic_coords;
      // [nb_cubics * 20]

   double *Quartic_coords;
      // [nb_quartics * 35]

   double *Quintic_coords;
      // [nb_quintics * 56]

   double *Octic_coords;
      // [nb_quartics * 165]

   int *Nb_face_points; // [nb_faces]
   int **Face_points; // [nb_faces]

public:


   std::vector<std::pair<int, std::string> > Labels;
```

```
double line_radius;

int nb_lines;

int nb_edges;

int nb_points;

int nb_planes;

int nb_quadrics;

int nb_cubics;

int nb_quartics;

int nb_quintics;

int nb_octics;

int nb_faces;

int nb_groups;
std::vector<std::vector<int> > group_of_things;

std::vector<int> animated_groups;

std::vector<drawable_set_of_objects> Drawables;



void *extra_data;

int f_has_affine_space;
int affine_space_q;
int affine_space_starting_point;

// scene_init.cpp:
int line6(double *x6);
int line(double x1, double x2, double x3,
   double y1, double y2, double y3);
int point(double x1, double x2, double x3);
int edge(int pt1, int pt2);
int plane(double x1, double x2, double x3, double a);
   // A plane is called a polynomial shape because
```

```
   // it is defined by a first order polynomial equation.
   // Given a plane: plane { <A, B, C>, D }
   // it can be represented by the equation
   // A*x + B*y + C*z - D*sqrt(A^2 + B^2 + C^2) = 0.
   // see http://www.povray.org/documentation/view/3.6.1/297/
int quadric(double *coeff10);
// povray ordering of monomials:
// http://www.povray.org/documentation/view/3.6.1/298/
// 1: x^2
// 2: xy
// 3: xz
// 4: x
// 5: y^2
// 6: yz
// 7: y
// 8: z^2
// 9: z
// 10: 1
int cubic(double *coeff20);
// povray ordering of monomials:
// http://www.povray.org/documentation/view/3.6.1/298/
// 1: x^3
// 2: x^2y
// 3: x^2z
// 4: x^2
// 5: xy^2
// 6: xyz
// 7: xy
// 8: xz^2
// 9: xz
// 10: x
// 11: y^3
// 12: y^2z
// 13: y^2
// 14: yz^2
// 15: yz
// 16: y
// 17: z^3
// 18: z^2
// 19: z
// 20: 1
int quartic(double *coeff35);
int quintic(double *coeff_56);
int octic(double *coeff_165);
int face(int *pts, int nb_pts);
int face3(int pt1, int pt2, int pt3);
int face4(int pt1, int pt2, int pt3, int pt4);
```

```
int face5(int pt1, int pt2, int pt3, int pt4, int pt5);



int line_pt_and_dir(double *x6, double rad, int verbose_level);
int line_pt_and_dir_and_copy_points(
      double *x6, double rad, int verbose_level);
int line_through_two_pts(double *x6, double rad);
int line_after_recentering(double x1, double x2, double x3,
   double y1, double y2, double y3, double rad);
int line_through_two_points(int pt1, int pt2,
   double rad);
int plane_through_three_points(int pt1, int pt2, int pt3);
int quadric_through_three_lines(int line_idx1,
   int line_idx2, int line_idx3, int verbose_level);
int cubic_in_orbiter_ordering(double *coeff);
void deformation_of_cubic_lex(int nb_frames,
      double angle_start, double angle_max, double angle_min,
      double *coeff1, double *coeff2,
      int verbose_level);
int cubic_Goursat_ABC(double A, double B, double C);
int line_extended(double x1, double x2, double x3,
   double y1, double y2, double y3,
   double r);


scene();
~scene();
double label(int idx, std::string &txt);
double point_coords(int idx, int j);
double line_coords(int idx, int j);
double plane_coords(int idx, int j);
double cubic_coords(int idx, int j);
double quadric_coords(int idx, int j);
int edge_points(int idx, int j);
void print_point_coords(int idx);
double point_distance_euclidean(int pt_idx, double *y);
double point_distance_from_origin(int pt_idx);
double distance_euclidean_point_to_point(
      int pt1_idx, int pt2_idx);
void init(int verbose_level);
scene *transformed_copy(double *A4, double *A4_inv,
   double rad, int verbose_level);
void print();
void transform_lines(scene *S, double *A4, double *A4_inv,
   double rad, int verbose_level);
void copy_edges(scene *S, double *A4, double *A4_inv,
```

```
   int verbose_level);
void transform_points(scene *S, double *A4, double *A4_inv,
   int verbose_level);
void transform_planes(scene *S, double *A4, double *A4_inv,
   int verbose_level);
void transform_quadrics(scene *S, double *A4, double *A4_inv,
   int verbose_level);
void transform_cubics(scene *S, double *A4, double *A4_inv,
   int verbose_level);
void transform_quartics(scene *S, double *A4, double *A4_inv,
   int verbose_level);
void transform_quintics(scene *S, double *A4, double *A4_inv,
   int verbose_level);
void copy_faces(scene *S, double *A4, double *A4_inv,
   int verbose_level);
void points(double *Coords, int nb_points);
int point_center_of_mass_of_face(int face_idx);
int point_center_of_mass_of_edge(int edge_idx);
int point_center_of_mass(int *Pt_idx, int nb_pts);
int triangle(int line1, int line2, int line3, int verbose_level);
int point_as_intersection_of_two_lines(int line1, int line2);
int plane_from_dual_coordinates(double *x4);
void draw_lines_with_selection(int *selection, int nb_select,
      std::string &options, std::ostream &ost);
void draw_line_with_selection(int line_idx,
      std::string &options, std::ostream &ost);
void draw_lines_cij_with_selection(int *selection, int nb_select,
      std::ostream &ost);
void draw_lines_cij(std::ostream &ost);
void draw_lines_cij_with_offset(int offset,
      int number_of_lines, std::ostream &ost);
void draw_lines_ai_with_selection(int *selection, int nb_select,
      std::ostream &ost);
void draw_lines_ai(std::ostream &ost);
void draw_lines_ai_with_offset(int offset, std::ostream &ost);
void draw_lines_bj_with_selection(int *selection, int nb_select,
      std::ostream &ost);
void draw_lines_bj(std::ostream &ost);
void draw_lines_bj_with_offset(int offset, std::ostream &ost);
void draw_edges_with_selection(int *selection, int nb_select,
      std::string &options, std::ostream &ost);
void draw_faces_with_selection(int *selection, int nb_select,
   double thickness_half, std::string &options, std::ostream &ost);
void draw_face(int idx, double thickness_half, std::string &options,
      std::ostream &ost);
void draw_planes_with_selection(int *selection, int nb_select,
      std::string &options, std::ostream &ost);
```

```
void draw_plane(int idx, std::string &options, std::ostream &ost);
void draw_points_with_selection(int *selection, int nb_select,
   double rad, std::string &options, std::ostream &ost);
void draw_cubic_with_selection(int *selection, int nb_select,
      std::string &options, std::ostream &ost);
void draw_quartic_with_selection(int *selection, int nb_select,
      std::string &options, std::ostream &ost);
void draw_quintic_with_selection(int *selection, int nb_select,
      std::string &options, std::ostream &ost);
void draw_octic_with_selection(int *selection, int nb_select,
      std::string &options, std::ostream &ost);
void draw_quadric_with_selection(int *selection, int nb_select,
      std::string &options, std::ostream &ost);
void draw_quadric_clipped_by_plane(int quadric_idx, int plane_idx,
      std::string &options, std::ostream &ost);
void draw_line_clipped_by_plane(int line_idx, int plane_idx,
      std::string &options, std::ostream &ost);
int intersect_line_and_plane(int line_idx, int plane_idx,
   int &intersection_point_idx,
   int verbose_level);
int intersect_line_and_line(int line1_idx, int line2_idx,
   double &lambda,
   int verbose_level);
void map_a_line(int line1, int line2,
   int plane_idx, int line_idx, double spread,
   int nb_pts,
   int *New_line_idx, int &nb_new_lines,
   int *New_pt_idx, int &nb_new_points, int verbose_level);
int map_a_point(int line1, int line2,
   int plane_idx, double pt_in[3],
   int &new_line_idx, int &new_pt_idx,
   int verbose_level);
void fourD_cube(double rad_desired);
void rescale(int first_pt_idx, double rad_desired);
double euclidean_distance(int pt1, int pt2);
double distance_from_origin(int pt);
void fourD_cube_edges(int first_pt_idx);
void hypercube(int n, double rad_desired);
void Dodecahedron_points();
void Dodecahedron_edges(int first_pt_idx);
void Dodecahedron_planes(int first_pt_idx);
void tritangent_planes();

// Clebsch version 1:
void clebsch_cubic();
void clebsch_cubic_lines_a();
void clebsch_cubic_lines_b();
```

```
   void clebsch_cubic_lines_cij();
   void Clebsch_Eckardt_points();

   // Clebsch version 2:
   void clebsch_cubic_version2();
   void clebsch_cubic_version2_Hessian();
   void clebsch_cubic_version2_lines_a();
   void clebsch_cubic_version2_lines_b();
   void clebsch_cubic_version2_lines_c();

   double distance_between_two_points(int pt1, int pt2);
   void create_five_plus_one();
   void create_Clebsch_surface(int verbose_level);
   // 1 cubic, 27 lines, 7 Eckardt points
   void create_Hilbert_Cohn_Vossen_surface(int verbose_level);
      // 1 cubic, 27 lines, 54 points, 45 planes
   void create_Hilbert_model(int verbose_level);
   void create_Cayleys_nodal_cubic(int verbose_level);
   void create_Hilbert_cube(int verbose_level);
   void create_cube(int verbose_level);
   void create_cube_and_tetrahedra(int verbose_level);
   void create_affine_space(int q, int verbose_level);
   //void create_surface_13_1(int verbose_level);
   void create_Eckardt_surface(int N, int verbose_level);
   void create_E4_surface(int N, int verbose_level);
   void create_twisted_cubic(int N, int verbose_level);
   void create_triangulation_of_cube(int N, int verbose_level);
   void print_a_line(int line_idx);
   void print_a_plane(int plane_idx);
   void print_a_face(int face_idx);
   void read_obj_file(std::string &fname, int verbose_level);
   void add_a_group_of_things(int *Idx, int sz, int verbose_level);
   void create_regulus(int idx, int nb_lines, int verbose_level);
   void clipping_by_cylinder(int line_idx, double r, std::ostream &ost);
   int scan1(int argc, std::string *argv, int &i, int verbose_level);
   int scan2(int argc, std::string *argv, int &i, int verbose_level);
   int read_scene_objects(int argc, std::string *argv,
         int i0, int verbose_level);
};


// ##############################################################################
// tree.cpp
// ##############################################################################


//! a data structure for trees
```

```
class tree {

public:

   tree_node *root;

   int nb_nodes;
   int max_depth;
   int *f_node_select;

   int *path;

   int f_count_leaves;
   int leaf_count;

   tree();
   ~tree();
   void init(graphics::tree_draw_options *Tree_draw_options,
         int xmax, int ymax, int verbose_level);
   void draw(std::string &fname,
         graphics::tree_draw_options *Tree_draw_options,
         layered_graph_draw_options *Opt,
         int verbose_level);
   void draw_preprocess(std::string &fname,
         graphics::tree_draw_options *Tree_draw_options,
         layered_graph_draw_options *Opt,
         int verbose_level);
   void circle_center_and_radii(int xmax, int ymax, int max_depth,
      int &x0, int &y0, int *&rad);
   void compute_DFS_ranks(int &nb_nodes, int verbose_level);
};

// ##############################################################################
// tree_draw_options.cpp
// ##############################################################################


//! options for drawing a tree


class tree_draw_options {

public:

   int f_file;
```

```
   std::string file_name;

   int f_restrict;
   int restrict_excluded_color;

   int f_select_path;
   std::string select_path_text;

   int f_has_draw_vertex_callback;
   void (*draw_vertex_callback)(tree *T,
      mp_graphics *G, int *v, int layer, tree_node *N,
      int x, int y, int dx, int dy);


   tree_draw_options();
   ~tree_draw_options();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// tree_node.cpp
// ##############################################################################


//! part of the data structure tree


class tree_node {

public:
   tree_node *parent;
   int depth;

   int f_value;
   int value;

   int f_has_color;
   int color;

   std::string label;

   int nb_children;
```

```
   tree_node **children;

   int weight;
   int placement_x;
   int placement_y;
   int width;

   int DFS_rank;

   tree_node();
   ~tree_node();
   void init(int depth, tree_node *parent, int f_value, int value,
      int f_has_color, int color, std::string &label,
      int verbose_level);
   void print_path();
   void print_depth_first();
   void compute_DFS_rank(int &rk);
   int find_node(int &DFS_rk, int *path, int sz, int verbose_level);
   int find_node_and_path(
         std::vector<int> &Rk, int *path,
         int sz, int verbose_level);
   void get_coordinates(int &idx, int *coord_xy);
   void get_coordinates_and_width(int &idx, int *coord_xyw);
   void calc_weight();
   void place_xy(int left, int right, int ymax, int max_depth);
   void place_on_circle(int xmax, int ymax, int max_depth);
   void add_node(int l, int depth, int *path,
         int color, std::string &label,
      int verbose_level);
   int find_child(int val);
   void get_values(int *v, int verbose_level);
   void draw_edges(mp_graphics &G,
         tree_draw_options *Tree_draw_options,
         layered_graph_draw_options *Opt,
      int f_has_parent, int parent_x, int parent_y, int max_depth,
      tree *T, int verbose_level);
   void draw_vertices(mp_graphics &G,
         tree_draw_options *Tree_draw_options,
         layered_graph_draw_options *Opt,
      int f_has_parent, int parent_x, int parent_y, int max_depth,
      tree *T, int verbose_level);
   void draw_sideways(mp_graphics &G, int f_circletext, int f_i,
      int f_has_parent, int parent_x, int parent_y,
      int max_depth, int f_edge_labels);
   int calc_y_coordinate(int ymax, int l, int max_depth);

};
```

```
// ##############################################################################
// video_draw_options.cpp:
// ##############################################################################


//! options for povray videos


class video_draw_options {
public:

   int f_rotate;
   int rotation_axis_type;
      // 1 = 1,1,1
      // 2 = 0,0,1
      // 3 = custom
   double rotation_axis_custom[3];
   int boundary_type;
      // 1 = sphere
      // 2 = box

   int f_has_global_picture_scale;
   double global_picture_scale;

   int f_has_font_size;
   int font_size;

   int f_has_stroke_width;
   int stroke_width;



   int f_W;
   int W;
   int f_H;
   int H;

   int f_default_angle; // = FALSE;
   int default_angle; // = 22;

   int f_clipping_radius; // = TRUE;
   double clipping_radius; // = 0.9;


   int nb_clipping;
```

```
int clipping_round[1000];
double clipping_value[1000];

int nb_camera;
int camera_round[1000];
double camera_sky[1000 * 3];
double camera_location[1000 * 3];
double camera_look_at[1000 * 3];

int nb_zoom;
int zoom_round[1000];
int zoom_start[1000];
int zoom_end[1000];
double zoom_clipping_start[1000];
double zoom_clipping_end[1000];

int nb_zoom_sequence;
int zoom_sequence_round[1000];
std::string zoom_sequence_text[1000];

int nb_pan;
int pan_round[1000];
int pan_f_reverse[1000];
double pan_from[1000 * 3];
double pan_to[1000 * 3];
double pan_center[1000 * 3];

int nb_no_background;
int no_background_round[1000];

int nb_no_bottom_plane;
int no_bottom_plane_round[1000];

int cnt_nb_frames;
int nb_frames_round[1000];
int nb_frames_value[1000];

int nb_round_text;
int round_text_round[1000];
int round_text_sustain[1000];
std::string round_text_text[1000];

int nb_label;
int label_round[1000];
int label_start[1000];
int label_sustain[1000];
std::string label_gravity[1000];
```

```
   std::string label_text[1000];

   int nb_latex_label;
   int latex_label_round[1000];
   int latex_label_start[1000];
   int latex_label_sustain[1000];
   std::string latex_extras_for_praeamble[1000];
   std::string latex_label_gravity[1000];
   std::string latex_label_text[1000];
   int latex_f_label_has_been_prepared[1000];
   std::string latex_fname_base[1000];


   int nb_picture;
   int picture_round[1000];
   double picture_scale[1000];
   std::string picture_fname[1000];
   std::string picture_options[1000];

   int latex_file_count;
   int f_omit_bottom_plane;

   double sky[3];
   double location[3];
   int f_look_at;
   double look_at[3];

   int f_scale_factor;
   double scale_factor;

   int f_line_radius;
   double line_radius;

   video_draw_options();
   ~video_draw_options();
   int read_arguments(
         int argc, std::string *argv,
         int verbose_level);
   void print();
};




}}}
```

```
#endif /* ORBITER_SRC_LIB_FOUNDATIONS_GRAPHICS_GRAPHICS_H_ */
```

3.15 Knowledge Base

```
/*
 * knowledge_base.h
 *
 *  Created on: May 20, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_FOUNDATIONS_KNOWLEDGE_BASE_KNOWLEDGE_BASE_H_
#define SRC_LIB_FOUNDATIONS_KNOWLEDGE_BASE_KNOWLEDGE_BASE_H_


namespace orbiter {
namespace layer1_foundations {
namespace knowledge_base {




// ##############################################################################
// knowledge_base.cpp:
// ##############################################################################

//! provides access to pre-computed combinatorial data in encoded form


class knowledge_base {
public:
   knowledge_base();
   ~knowledge_base();


   // the index i is zero-based:


   int quartic_curves_nb_reps(int q);
   int *quartic_curves_representative(int q, int i);
   long int *quartic_curves_bitangents(int q, int i);
   void quartic_curves_stab_gens(int q, int i,
         int *&data, int &nb_gens,
         int &data_size, std::string &stab_order_str);


   int cubic_surface_nb_reps(int q);
   int *cubic_surface_representative(int q, int i);
   void cubic_surface_stab_gens(int q, int i, int *&data, int &nb_gens,
      int &data_size, std::string &stab_order_str);
```

```
int cubic_surface_nb_Eckardt_points(int q, int i);
long int *cubic_surface_Lines(int q, int i);

int hyperoval_nb_reps(int q);
int *hyperoval_representative(int q, int i);
void hyperoval_gens(int q, int i, int *&data, int &nb_gens,
   int &data_size, std::string &stab_order_str);


int DH_nb_reps(int k, int n);
long int *DH_representative(int k, int n, int i);
void DH_stab_gens(int k, int n, int i, int *&data, int &nb_gens,
   int &data_size, std::string &stab_order_str);

int Spread_nb_reps(int q, int k);
long int *Spread_representative(int q, int k, int i, int &sz);
void Spread_stab_gens(int q, int k, int i, int *&data, int &nb_gens,
   int &data_size, std::string &stab_order_str);

int BLT_nb_reps(int q);
long int *BLT_representative(int q, int no);
void BLT_stab_gens(int q, int no, int *&data, int &nb_gens,
   int &data_size, std::string &stab_order_str);

void override_polynomial_subfield(std::string &poly, int q);
void override_polynomial_extension_field(std::string &poly, int q);

void get_projective_plane_list_of_lines(int *&list_of_lines,
      int &order, int &nb_lines, int &line_size,
      const char *label, int verbose_level);

int tensor_orbits_nb_reps(int n);
long int *tensor_orbits_rep(int n, int idx);

void retrieve_BLT_set_from_database_embedded(
      orthogonal_geometry::quadratic_form *Quadratic_form,
      int BLT_k,
      std::string &label_txt,
      std::string &label_tex,
      int &nb_pts, long int *&Pts,
      int verbose_level);
void retrieve_BLT_set_from_database(
      orthogonal_geometry::quadratic_form *Quadratic_form,
      int f_embedded,
      int BLT_k,
      std::string &label_txt,
      std::string &label_tex,
```

```
        int &nb_pts, long int *&Pts,
        int verbose_level);

   // finitefield_tables.cpp:
   void get_primitive_polynomial(std::string &poly, int p, int e, int verbose_leve
l);

};


}}}


#endif /* SRC_LIB_FOUNDATIONS_KNOWLEDGE_BASE_KNOWLEDGE_BASE_H_ */
```

3.16 Linear Algebra

```
/*
 * linear_algebra.h
 *
 *  Created on: Jan 10, 2022
 *      Author: betten
 */

#ifndef SRC_LIB_FOUNDATIONS_LINEAR_ALGEBRA_LINEAR_ALGEBRA_H_
#define SRC_LIB_FOUNDATIONS_LINEAR_ALGEBRA_LINEAR_ALGEBRA_H_


namespace orbiter {
namespace layer1_foundations {
namespace linear_algebra {


// ##############################################################################
// linear_algebra_global.cpp:
// ##############################################################################

//! catch all class for functions related to linear algebra

class linear_algebra_global {
public:
   linear_algebra_global();
   ~linear_algebra_global();
   void Berlekamp_matrix(
         field_theory::finite_field *F,
         std::string &Berlekamp_matrix_label,
         int verbose_level);
   void compute_normal_basis(
         field_theory::finite_field *F,
         int d, int verbose_level);
   void do_nullspace(
         field_theory::finite_field *F,
         int *M, int m, int n,
         int f_normalize_from_the_left, int f_normalize_from_the_right,
         int verbose_level);
   void do_RREF(
         field_theory::finite_field *F,
         int *M, int m, int n,
         int f_normalize_from_the_left, int f_normalize_from_the_right,
         int verbose_level);
   void RREF_demo(
         field_theory::finite_field *F,
         int *A, int m, int n, int verbose_level);
```

```
   void RREF_with_steps_latex(
         field_theory::finite_field *F,
         std::ostream &ost, int *A, int m, int n, int verbose_level);

};



// ##############################################################################
// linear_algebra.cpp:
// ##############################################################################

//! linear algebra over a finite field

class linear_algebra {
public:
   field_theory::finite_field *F;


   linear_algebra();
   ~linear_algebra();
   void init(field_theory::finite_field *F, int verbose_level);


   void copy_matrix(int *A, int *B, int ma, int na);
   void reverse_matrix(int *A, int *B, int ma, int na);
   void identity_matrix(int *A, int n);
   int is_identity_matrix(int *A, int n);
   int is_diagonal_matrix(int *A, int n);
   int is_scalar_multiple_of_identity_matrix(int *A,
      int n, int &scalar);
   void diagonal_matrix(int *A, int n, int alpha);
   void matrix_minor(int f_semilinear, int *A,
      int *B, int n, int f, int l);
      // initializes B as the l x l minor of A
      // (which is n x n) starting from row f.
   void mult_vector_from_the_left(int *v, int *A,
      int *vA, int m, int n);
      // v[m], A[m][n], vA[n]
   void mult_vector_from_the_right(int *A, int *v,
      int *Av, int m, int n);
      // A[m][n], v[n], Av[m]

   void mult_matrix_matrix(int *A, int *B,
      int *C, int m, int n, int o, int verbose_level);
      // matrix multiplication C := A * B,
      // where A is m x n and B is n x o, so that C is m by o
```

```
void semilinear_matrix_mult(int *A, int *B, int *AB, int n);
   // (A,f1) * (B,f2) = (A*B^{\varphi^{-f1}},f1+f2)
void semilinear_matrix_mult_memory_given(int *A, int *B,
   int *AB, int *tmp_B, int n, int verbose_level);
   // (A,f1) * (B,f2) = (A*B^{\varphi^{-f1}},f1+f2)
void matrix_mult_affine(int *A, int *B, int *AB,
   int n, int verbose_level);
void semilinear_matrix_mult_affine(int *A, int *B, int *AB, int n);
int matrix_determinant(int *A, int n, int verbose_level);
void matrix_inverse(int *A, int *Ainv, int n, int verbose_level);
void matrix_invert(int *A, int *Tmp,
   int *Tmp_basecols, int *Ainv, int n, int verbose_level);
   // Tmp points to n * n + 1 int´s
   // Tmp_basecols points to n int´s
void semilinear_matrix_invert(int *A, int *Tmp,
   int *Tmp_basecols, int *Ainv, int n, int verbose_level);
   // Tmp points to n * n + 1 int´s
   // Tmp_basecols points to n int´s
void semilinear_matrix_invert_affine(int *A, int *Tmp,
   int *Tmp_basecols, int *Ainv, int n, int verbose_level);
   // Tmp points to n * n + 1 int´s
   // Tmp_basecols points to n int´s
void matrix_invert_affine(int *A, int *Tmp, int *Tmp_basecols,
   int *Ainv, int n, int verbose_level);
   // Tmp points to n * n + 1 int´s
   // Tmp_basecols points to n int´s
void projective_action_from_the_right(int f_semilinear,
   int *v, int *A, int *vA, int n, int verbose_level);
   // vA = (v * A)^{p^f}  if f_semilinear (where f = A[n *  n]),
   // vA = v * A otherwise
void general_linear_action_from_the_right(int f_semilinear,
   int *v, int *A, int *vA, int n, int verbose_level);
void semilinear_action_from_the_right(int *v,
   int *A, int *vA, int n);
   // vA = (v * A)^{p^f}  (where f = A[n *  n])
void semilinear_action_from_the_left(int *A,
   int *v, int *Av, int n);
   // Av = (A * v)^{p^f}
void affine_action_from_the_right(int f_semilinear,
   int *v, int *A, int *vA, int n);
   // vA = (v * A)^{p^f} + b
void zero_vector(int *A, int m);
void all_one_vector(int *A, int m);
void support(int *A, int m, int *&support, int &size);
void characteristic_vector(int *A, int m, int *set, int size);
int is_zero_vector(int *A, int m);
void add_vector(int *A, int *B, int *C, int m);
```

```
void linear_combination_of_vectors(
      int a, int *A, int b, int *B, int *C, int len);
void linear_combination_of_three_vectors(
      int a, int *A, int b, int *B, int c, int *C, int *D, int len);
void negate_vector(int *A, int *B, int m);
void negate_vector_in_place(int *A, int m);
void scalar_multiply_vector_in_place(int c, int *A, int m);
void vector_frobenius_power_in_place(int *A, int m, int f);
int dot_product(int len, int *v, int *w);
void transpose_matrix(int *A, int *At, int ma, int na);
void transpose_matrix_in_place(int *A, int m);
void invert_matrix(int *A, int *A_inv, int n, int verbose_level);
void invert_matrix_memory_given(int *A, int *A_inv, int n,
      int *tmp_A, int *tmp_basecols, int verbose_level);
void transform_form_matrix(int *A, int *Gram,
   int *new_Gram, int d, int verbose_level);
   // computes new_Gram = A * Gram * A^\top
int rank_of_matrix(int *A, int m, int verbose_level);
int rank_of_matrix_memory_given(int *A,
   int m, int *B, int *base_cols, int verbose_level);
int rank_of_rectangular_matrix(int *A,
   int m, int n, int verbose_level);
int rank_of_rectangular_matrix_memory_given(int *A,
   int m, int n, int *B, int *base_cols, int f_complete,
   int verbose_level);
int rank_and_basecols(int *A, int m,
   int *base_cols, int verbose_level);
void Gauss_step(int *v1, int *v2, int len,
   int idx, int verbose_level);
   // afterwards: v2[idx] = 0
   // and v1,v2 span the same space as before
   // v1 is not changed if v1[idx] is nonzero
void Gauss_step_make_pivot_one(int *v1, int *v2,
   int len, int idx, int verbose_level);
   // afterwards:  v1,v2 span the same space as before
   // v2[idx] = 0, v1[idx] = 1,
void extend_basis(int m, int n, int *Basis, int verbose_level);
int base_cols_and_embedding(int m, int n, int *A,
   int *base_cols, int *embedding, int verbose_level);
   // returns the rank rk of the matrix.
   // It also computes base_cols[rk] and embedding[m - rk]
   // It leaves A unchanged
int Gauss_easy(int *A, int m, int n);
   // returns the rank
int Gauss_easy_memory_given(int *A, int m, int n, int *base_cols);
int Gauss_simple(int *A, int m, int n,
   int *base_cols, int verbose_level);
```

```
   // returns the rank which is the
   // number of entries in base_cols
void kernel_columns(int n, int nb_base_cols,
   int *base_cols, int *kernel_cols);
void matrix_get_kernel_as_int_matrix(int *M, int m, int n,
   int *base_cols, int nb_base_cols,
   data_structures::int_matrix *kernel, int verbose_level);
void matrix_get_kernel(int *M, int m, int n,
   int *base_cols, int nb_base_cols,
   int &kernel_m, int &kernel_n, int *kernel, int verbose_level);
   // kernel[n * (n - nb_base_cols)]
int perp(int n, int k, int *A, int *Gram, int verbose_level);
int RREF_and_kernel(int n, int k, int *A, int verbose_level);
int perp_standard(int n, int k, int *A, int verbose_level);
int perp_standard_with_temporary_data(int n, int k, int *A,
   int *B, int *K, int *base_cols,
   int verbose_level);
int intersect_subspaces(int n, int k1, int *A, int k2, int *B,
   int &k3, int *intersection, int verbose_level);
int n_choose_k_mod_p(int n, int k, int verbose_level);
void Dickson_polynomial(int *map, int *coeffs);
   // compute the coefficients of a degree q-1 polynomial
   // which interpolates a given map
   // from F_q to F_q
void projective_action_on_columns_from_the_left(int *A,
   int *M, int m, int n, int *perm, int verbose_level);
int evaluate_bilinear_form(int n, int *v1, int *v2, int *Gram);
int evaluate_standard_hyperbolic_bilinear_form(int n,
   int *v1, int *v2);
int evaluate_quadratic_form(int n, int nb_terms,
   int *i, int *j, int *coeff, int *x);
void find_singular_vector_brute_force(int n, int form_nb_terms,
   int *form_i, int *form_j, int *form_coeff, int *Gram,
   int *vec, int verbose_level);
void find_singular_vector(int n, int form_nb_terms,
   int *form_i, int *form_j, int *form_coeff, int *Gram,
   int *vec, int verbose_level);
void complete_hyperbolic_pair(int n, int form_nb_terms,
   int *form_i, int *form_j, int *form_coeff, int *Gram,
   int *vec1, int *vec2, int verbose_level);
void find_hyperbolic_pair(int n, int form_nb_terms,
   int *form_i, int *form_j, int *form_coeff, int *Gram,
   int *vec1, int *vec2, int verbose_level);
void restrict_quadratic_form_list_coding(int k,
   int n, int *basis,
   int form_nb_terms,
   int *form_i, int *form_j, int *form_coeff,
```

```
      int &restricted_form_nb_terms,
      int *&restricted_form_i, int *&restricted_form_j,
      int *&restricted_form_coeff,
      int verbose_level);
   void restrict_quadratic_form(int k, int n, int *basis,
      int *C, int *D, int verbose_level);
   int compare_subspaces_ranked(int *set1, int *set2, int size,
      int vector_space_dimension, int verbose_level);
      // Compares the span of two sets of vectors.
      // returns 0 if equal, 1 if not
      // (this is so that it matches to the result
      // of a compare function)
   int compare_subspaces_ranked_with_unrank_function(
      int *set1, int *set2, int size,
      int vector_space_dimension,
      void (*unrank_point_func)(int *v, int rk, void *data),
      void *rank_point_data,
      int verbose_level);
   int Gauss_canonical_form_ranked(int *set1, int *set2, int size,
      int vector_space_dimension, int verbose_level);
      // Computes the Gauss canonical form
      // for the generating set in set1.
      // The result is written to set2.
      // Returns the rank of the span of the elements in set1.
   int lexleast_canonical_form_ranked(int *set1, int *set2, int size,
      int vector_space_dimension, int verbose_level);
      // Computes the lexleast generating set the subspace
      // spanned by the elements in set1.
      // The result is written to set2.
      // Returns the rank of the span of the elements in set1.
   void exterior_square(int *An, int *An2, int n, int verbose_level);
   void lift_to_Klein_quadric(int *A4, int *A6, int verbose_level);


   // linear_algebra2.cpp

   void get_coefficients_in_linear_combination(
      int k, int n, int *basis_of_subspace,
      int *input_vector, int *coefficients, int verbose_level);
      // basis[k * n]
      // coefficients[k]
      // input_vector[n] is the input vector.
      // At the end, coefficients[k] are the coefficients of the linear combination

      // which expresses input_vector[n] in terms of the given basis of the subspac
e.
   void reduce_mod_subspace_and_get_coefficient_vector(
```

```
   int k, int len, int *basis, int *base_cols,
   int *v, int *coefficients, int verbose_level);
void reduce_mod_subspace(int k,
   int len, int *basis, int *base_cols,
   int *v, int verbose_level);
int is_contained_in_subspace(int k,
   int len, int *basis, int *base_cols,
   int *v, int verbose_level);
int is_subspace(int d, int dim_U, int *Basis_U, int dim_V,
   int *Basis_V, int verbose_level);
void Kronecker_product(int *A, int *B, int n, int *AB);
void Kronecker_product_square_but_arbitrary(int *A, int *B,
   int na, int nb, int *AB, int &N, int verbose_level);
int dependency(int d, int *v, int *A, int m, int *rho,
   int verbose_level);
   // Lueneburg~\cite{Lueneburg87a} p. 104.
   // A is a matrix of size d + 1 times d
   // v[d]
   // rho is a column permutation of degree d
void order_ideal_generator(int d, int idx, int *mue, int &mue_deg,
   int *A, int *Frobenius,
   int verbose_level);
   // Lueneburg~\cite{Lueneburg87a} p. 105.
   // Frobenius is a matrix of size d x d
   // A is (d + 1) x d
   // mue[d + 1]
void span_cyclic_module(int *A, int *v, int n, int *Mtx,
   int verbose_level);
void random_invertible_matrix(int *M, int k, int verbose_level);
void adjust_basis(int *V, int *U, int n, int k, int d,
   int verbose_level);
void choose_vector_in_here_but_not_in_here_column_spaces(
   data_structures::int_matrix *V, data_structures::int_matrix *W,
   int *v, int verbose_level);
void choose_vector_in_here_but_not_in_here_or_here_column_spaces(
      data_structures::int_matrix *V, data_structures::int_matrix *W1,
      data_structures::int_matrix *W2, int *v,
   int verbose_level);
int
choose_vector_in_here_but_not_in_here_or_here_column_spaces_coset(
   int &coset,
   data_structures::int_matrix *V,
   data_structures::int_matrix *W1,
   data_structures::int_matrix *W2, int *v,
   int verbose_level);
void vector_add_apply(int *v, int *w, int c, int n);
void vector_add_apply_with_stride(int *v, int *w, int stride,
```

```
   int c, int n);
int test_if_commute(int *A, int *B, int k, int verbose_level);
void unrank_point_in_PG(int *v, int len, int rk);
   // len is the length of the vector,
   // not the projective dimension
int rank_point_in_PG(int *v, int len);
int nb_points_in_PG(int n);
   // n is projective dimension
void Borel_decomposition(int n, int *M, int *B1, int *B2,
   int *pivots, int verbose_level);
void map_to_standard_frame(int d, int *A,
   int *Transform, int verbose_level);
   // d = vector space dimension
   // maps d + 1 points to the frame
   // e_1, e_2, ..., e_d, e_1+e_2+..+e_d
   // A is (d + 1) x d
   // Transform is d x d
void map_frame_to_frame_with_permutation(int d, int *A,
   int *perm, int *B, int *Transform, int verbose_level);
void map_points_to_points_projectively(int d, int k,
   int *A, int *B, int *Transform,
   int &nb_maps, int verbose_level);
   // A and B are (d + k + 1) x d
   // Transform is d x d
   // returns TRUE if a map exists
int BallChowdhury_matrix_entry(int *Coord, int *C,
   int *U, int k, int sz_U,
   int *T, int verbose_level);
void cubic_surface_family_24_generators(int f_with_normalizer,
   int f_semilinear,
   int *&gens, int &nb_gens, int &data_size,
   int &group_order, int verbose_level);
void cubic_surface_family_G13_generators(
      int a,
      int *&gens, int &nb_gens, int &data_size,
      int &group_order, int verbose_level);
void cubic_surface_family_F13_generators(
   int a,
   int *&gens, int &nb_gens, int &data_size,
   int &group_order, int verbose_level);
int is_unit_vector(int *v, int len, int k);
void make_Fourier_matrices(
      int omega, int k, int *N, int **A, int **Av,
      int *Omega, int verbose_level);

// linear_algebra3.cpp
int evaluate_conic_form(int *six_coeffs, int *v3);
```

```
int evaluate_quadric_form_in_PG_three(int *ten_coeffs, int *v4);
int Pluecker_12(int *x4, int *y4);
int Pluecker_21(int *x4, int *y4);
int Pluecker_13(int *x4, int *y4);
int Pluecker_31(int *x4, int *y4);
int Pluecker_14(int *x4, int *y4);
int Pluecker_41(int *x4, int *y4);
int Pluecker_23(int *x4, int *y4);
int Pluecker_32(int *x4, int *y4);
int Pluecker_24(int *x4, int *y4);
int Pluecker_42(int *x4, int *y4);
int Pluecker_34(int *x4, int *y4);
int Pluecker_43(int *x4, int *y4);
int Pluecker_ij(int i, int j, int *x4, int *y4);
int evaluate_symplectic_form(int len, int *x, int *y);
int evaluate_symmetric_form(int len, int *x, int *y);
int evaluate_quadratic_form_x0x3mx1x2(int *x);
void solve_y2py(int a, int *Y2, int &nb_sol);
void find_secant_points_wrt_x0x3mx1x2(
      int *Basis_line,
      int *Pts4, int &nb_pts, int verbose_level);
int is_totally_isotropic_wrt_symplectic_form(int k,
   int n, int *Basis);
int evaluate_monomial(int *monomial, int *variables, int nb_vars);



// linear_algebra_RREF.cpp

int Gauss_int(int *A, int f_special,
   int f_complete, int *base_cols,
   int f_P, int *P, int m, int n,
   int Pn, int verbose_level);
   // returns the rank which is the
   // number of entries in base_cols
   // A is m x n,
   // P is m x Pn (provided f_P is TRUE)
int Gauss_int_with_pivot_strategy(int *A,
   int f_special, int f_complete, int *pivot_perm,
   int m, int n,
   int (*find_pivot_function)(int *A, int m, int n, int r,
   int *pivot_perm, void *data),
   void *find_pivot_data,
   int verbose_level);
   // returns the rank which is the number of entries in pivots
   // A is a m x n matrix
int Gauss_int_with_given_pivots(int *A,
```

```
      int f_special, int f_complete, int *pivots, int nb_pivots,
      int m, int n,
      int verbose_level);
      // A is a m x n matrix
      // returns FALSE if pivot cannot be found at one of the steps
   int RREF_search_pivot(int *A, int m, int n,
         int &i, int &j, int *base_cols, int verbose_level);
   void RREF_make_pivot_one(int *A, int m, int n,
         int &i, int &j, int *base_cols, int verbose_level);
   void RREF_elimination_below(int *A, int m, int n,
         int &i, int &j, int *base_cols, int verbose_level);
   void RREF_elimination_above(int *A, int m, int n,
         int i, int *base_cols, int verbose_level);


};



// ##############################################################################
// representation_theory_domain.cpp:
// ##############################################################################

//! catch all class for representation theory

class representation_theory_domain {
public:

   field_theory::finite_field *F;

   representation_theory_domain();
   ~representation_theory_domain();
   void init(field_theory::finite_field *F, int verbose_level);
   void representing_matrix8_R(int *A,
      int q, int a, int b, int c, int d);
   void representing_matrix9_R(int *A,
      int q, int a, int b, int c, int d);
   void representing_matrix9_U(int *A,
      int a, int b, int c, int d, int beta);
   void representing_matrix8_U(int *A,
      int a, int b, int c, int d, int beta);
   void representing_matrix8_V(int *A, int beta);
   void representing_matrix9b(int *A, int beta);
   void representing_matrix8a(int *A,
      int a, int b, int c, int d, int beta);
   void representing_matrix8b(int *A, int beta);
   int Term1(int a1, int e1);
```

```
   int Term2(int a1, int a2, int e1, int e2);
   int Term3(int a1, int a2, int a3, int e1, int e2, int e3);
   int Term4(int a1, int a2, int a3, int a4, int e1, int e2, int e3,
      int e4);
   int Term5(int a1, int a2, int a3, int a4, int a5, int e1, int e2,
      int e3, int e4, int e5);
   int term1(int a1, int e1);
   int term2(int a1, int a2, int e1, int e2);
   int term3(int a1, int a2, int a3, int e1, int e2, int e3);
   int term4(int a1, int a2, int a3, int a4, int e1, int e2, int e3,
      int e4);
   int term5(int a1, int a2, int a3, int a4, int a5, int e1, int e2,
      int e3, int e4, int e5);
   int m_term(int q, int a1, int a2, int a3);
   int beta_trinomial(int q, int beta, int a1, int a2, int a3);
   int T3product2(int a1, int a2);
   int add(int a, int b);
   int add3(int a, int b, int c);
   int negate(int a);
   int twice(int a);
   int mult(int a, int b);
   int inverse(int a);
   int power(int a, int n);
   int T2(int a);
   int T3(int a);
   int N2(int a);
   int N3(int a);

};



}}}



#endif /* SRC_LIB_FOUNDATIONS_LINEAR_ALGEBRA_LINEAR_ALGEBRA_H_ */
```

3.17 Number Theory

```
/*
 * number_theory.h
 *
 *  Created on: Nov 2, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_FOUNDATIONS_NUMBER_THEORY_NUMBER_THEORY_H_
#define SRC_LIB_FOUNDATIONS_NUMBER_THEORY_NUMBER_THEORY_H_



namespace orbiter {
namespace layer1_foundations {
namespace number_theory {



// ##############################################################################
// cyclotomic_sets.cpp
// ##############################################################################


//! cyclotomic sets for cyclic codes



class cyclotomic_sets {
public:
   int n;
   int q;
   int m;
   int qm;

   int *Index;
   data_structures::set_of_sets *S;

   cyclotomic_sets();
   ~cyclotomic_sets();
   void init(
         field_theory::finite_field *F,
         int n, int verbose_level);
   void print();
   void print_latex(std::ostream &ost);
   void print_latex_with_selection(
         std::ostream &ost, int *Selection, int nb_sel);
```

```
};


// ##############################################################################
// elliptic_curve.cpp
// ##############################################################################



//! a fixed elliptic curve in Weierstrass form



class elliptic_curve {

public:
   int q;
   int p;
   int e;
   int b, c;
      // the equation of the curve is
      // Y^2 = X^3 + bX + c mod p
   int nb; // number of points
   int *T; // [nb * 3] point coordinates
      // the point at infinity is last
   int *A; // [nb * nb] addition table
   field_theory::finite_field *F;


   elliptic_curve();
   ~elliptic_curve();
   void init(
         field_theory::finite_field *F,
         int b, int c, int verbose_level);
   void compute_points(int verbose_level);
   void add_point_to_table(int x, int y, int z);
   int evaluate_RHS(int x);
      // evaluates x^3 + bx + c
   void print_points();
   void print_points_affine();
   void addition(
      int x1, int y1, int z1,
      int x2, int y2, int z2,
      int &x3, int &y3, int &z3, int verbose_level);
   void save_incidence_matrix(
         std::string &fname, int verbose_level);
```

```
   void draw_grid(
         std::string &fname,
         graphics::layered_graph_draw_options *Draw_options,
         int f_with_grid, int f_with_points, int point_density,
         int f_path, int start_idx, int nb_steps,
         int verbose_level);
   void draw_grid2(
         graphics::mp_graphics &G,
         int f_with_grid, int f_with_points, int point_density,
         int f_path, int start_idx, int nb_steps,
         int verbose_level);
   void make_affine_point(int x1, int x2, int x3,
      int &a, int &b, int verbose_level);
   void compute_addition_table(int verbose_level);
   void print_addition_table();
   int index_of_point(int x1, int x2, int x3);
   void latex_points_with_order(std::ostream &ost);
   void latex_order_of_all_points(std::ostream &ost);
   void order_of_all_points(std::vector<int> &Ord);
   int order_of_point(int i);
   void print_all_powers(int i);
};



// ##############################################################################
// number_theoretic_transform.cpp:
// ##############################################################################

//! Fourier transform over finite fields

class number_theoretic_transform {
public:

   int k;
   int q;

   std::string fname_code;

   field_theory::finite_field *F; // no ownership, do not destroy

   int *N;

   int alpha, omega;
   int gamma, minus_gamma, minus_one;
   int **A; // Fourier matrices for the positively wrapped convolution
   int **Av;
```

```
int **A_log;
int *Omega;


int **G;
int **D;
int **Dv;
int **T;
int **Tv;
int **P;

int *X, *Y, *Z;
int *X1, *X2;
int *Y1, *Y2;

int **Gr;
int **Dr;
int **Dvr;
int **Tr;
int **Tvr;
int **Pr;

int *Tmp1;
int *Tmp2;

int *bit_reversal;

int Q;
field_theory::finite_field *FQ;
int alphaQ;
int psi;
int *Psi_powers; // powers of psi


number_theoretic_transform();
~number_theoretic_transform();
void init(field_theory::finite_field *F,
      int k, int q, int verbose_level);
void write_code(std::string &fname_code,
      int verbose_level);
void write_code2(std::ostream &ost,
      int f_forward,
      int &nb_add, int &nb_negate, int &nb_mult,
      int verbose_level);
void write_code_header(std::ostream &ost,
      std::string &fname_code, int verbose_level);
void make_level(int s, int verbose_level);
```

```
   void paste(int **Xr, int **X, int s, int verbose_level);
   void make_G_matrix(int s, int verbose_level);
   void make_D_matrix(int s, int verbose_level);
   void make_T_matrix(int s, int verbose_level);
   void make_P_matrix(int s, int verbose_level);
   void multiply_matrix_stack(
         field_theory::finite_field *F, int **S,
         int nb, int sz, int *Result, int verbose_level);
};

// ##############################################################################
// number_theory_domain.cpp:
// ##############################################################################

//! basic number theoretic functions


class number_theory_domain {

public:
   number_theory_domain();
   ~number_theory_domain();
   long int mod(long int a, long int p);
   long int int_negate(long int a, long int p);
   long int power_mod(long int a, long int n, long int p);
   long int inverse_mod(long int a, long int p);
   long int mult_mod(long int a, long int b, long int p);
   long int add_mod(long int a, long int b, long int p);
   long int ab_over_c(long int a, long int b, long int c);
   long int int_abs(long int a);
   long int gcd_lint(long int m, long int n);
   void extended_gcd_int(int m, int n, int &g, int &u, int &v);
   void extended_gcd_lint(long int m, long int n,
         long int &g, long int &u, long int &v);
   long int gcd_with_key_in_latex(std::ostream &ost,
         long int a, long int b, int f_key, int verbose_level);
   int i_power_j_safe(int i, int j);
   long int i_power_j_lint_safe(int i, int j, int verbose_level);
   long int i_power_j_lint(long int i, long int j);
   int i_power_j(int i, int j);
   void do_eulerfunction_interval(
         long int n_min, long int n_max, int verbose_level);
   long int euler_function(long int n);
   long int moebius_function(long int n);
   long int order_mod_p(long int a, long int p);
   int int_log2(int n);
   int int_log10(int n);
```

```
int lint_log10(long int n);
int int_logq(int n, int q);
// returns the number of digits in base q representation
int lint_logq(long int n, int q);
int is_strict_prime_power(int q);
// assuming that q is a prime power, this function tests
// if q is a strict prime power
int is_prime(int p);
int is_prime_power(int q);
int is_prime_power(int q, int &p, int &h);
int smallest_primedivisor(int n);
//Computes the smallest prime dividing $n$.
//The algorithm is based on Lueneburg~\cite{Lueneburg87a}.
int sp_ge(int n, int p_min);
int factor_int(int a, int *&primes, int *&exponents);
int nb_prime_factors_counting_multiplicities(long int a);
int nb_distinct_prime_factors(long int a);
void factor_lint(
      long int a,
      std::vector<long int> &primes,
      std::vector<int> &exponents);
void factor_prime_power(int q, int &p, int &e);
long int primitive_root_randomized(long int p, int verbose_level);
long int primitive_root(long int p, int verbose_level);
int Legendre(long int a, long int p, int verbose_level);
int Jacobi(long int a, long int m, int verbose_level);
int Jacobi_with_key_in_latex(std::ostream &ost,
      long int a, long int m, int verbose_level);
int Legendre_with_key_in_latex(std::ostream &ost,
      long int a, long int m, int verbose_level);
//Computes the Legendre symbol $\left( \frac{a}{m} \right)$.
int ny2(long int x, long int &x1);
int ny_p(long int n, long int p);
//long int sqrt_mod_simple(long int a, long int p);
void print_factorization(
      int nb_primes, int *primes, int *exponents);
void print_longfactorization(
      int nb_primes,
      ring_theory::longinteger_object *primes,
      int *exponents);
void int_add_fractions(int at, int ab, int bt, int bb,
   int &ct, int &cb, int verbose_level);
void int_mult_fractions(int at, int ab, int bt, int bb,
   int &ct, int &cb, int verbose_level);
int choose_a_prime_greater_than(int lower_bound);
int choose_a_prime_in_interval(int lower_bound, int upper_bound);
int random_integer_greater_than(int n, int lower_bound);
```

```
int random_integer_in_interval(int lower_bound, int upper_bound);
int nb_primes_available();
int get_prime_from_table(int idx);
long int Chinese_Remainders(
      std::vector<long int> &Remainders,
      std::vector<long int> &Moduli,
      long int &M, int verbose_level);
long int ChineseRemainder2(long int a1, long int a2,
      long int p1, long int p2, int verbose_level);
void sieve(std::vector<int> &primes,
      int factorbase, int verbose_level);
void sieve_primes(std::vector<int> &v,
      int from, int to, int limit, int verbose_level);
int nb_primes(int n);
void cyclotomic_set(
      std::vector<int> &cyclotomic_set,
      int a, int q, int n, int verbose_level);
void elliptic_curve_addition(
      field_theory::finite_field *F,
      int b, int c,
   int x1, int x2, int x3,
   int y1, int y2, int y3,
   int &z1, int &z2, int &z3, int verbose_level);
void elliptic_curve_point_multiple(
      field_theory::finite_field *F,
      int b, int c, int n,
   int x1, int y1, int z1,
   int &x3, int &y3, int &z3,
   int verbose_level);
void elliptic_curve_point_multiple_with_log(
      field_theory::finite_field *F,
      int b, int c, int n,
   int x1, int y1, int z1,
   int &x3, int &y3, int &z3,
   int verbose_level);
int elliptic_curve_evaluate_RHS(
      field_theory::finite_field *F,
      int x, int b, int c);
void elliptic_curve_points(
      field_theory::finite_field *F,
      int b, int c, int &nb, int *&T,
      int verbose_level);
void elliptic_curve_all_point_multiples(
      field_theory::finite_field *F,
      int b, int c, int &order,
   int x1, int y1, int z1,
   std::vector<std::vector<int> > &Pts,
```

```
      int verbose_level);
   int elliptic_curve_discrete_log(
         field_theory::finite_field *F,
         int b, int c,
      int x1, int y1, int z1,
      int x3, int y3, int z3,
      int verbose_level);
   int eulers_totient_function(
         int n, int verbose_level);
   void do_jacobi(
         long int jacobi_top,
         long int jacobi_bottom, int verbose_level);
   void elliptic_curve_addition_table(
         geometry::projective_space *P2,
      int *A6, int *Pts, int nb_pts, int *&Table,
      int verbose_level);
   int elliptic_curve_addition(
         geometry::projective_space *P2,
      int *A6, int p1_rk, int p2_rk,
      int verbose_level);

};




}}}



#endif /* SRC_LIB_FOUNDATIONS_NUMBER_THEORY_NUMBER_THEORY_H_ */
```

3.18 Orbiter Kernel System

```
// orbiter_kernel_system.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005


#ifndef ORBITER_SRC_LIB_FOUNDATIONS_IO_AND_OS_IO_AND_OS_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_IO_AND_OS_IO_AND_OS_H_





namespace orbiter {
namespace layer1_foundations {
namespace orbiter_kernel_system {



// ##############################################################################
// create_file_description.cpp
// ##############################################################################


#define MAX_LINES 100

//! rules to create text files


class create_file_description {
public:
   int f_file_mask;
   std::string file_mask;
   int f_N;
   int N;
   int nb_lines;
   std::string lines[MAX_LINES];
   int f_line_numeric[MAX_LINES];
   int nb_final_lines;
   std::string final_lines[MAX_LINES];
   int f_command;
   std::string command;
```

```
   int f_repeat;
   int repeat_N;
   int repeat_start;
   int repeat_increment;
   std::string repeat_mask;
   int f_split;
   int split_m;
   int f_read_cases;
   std::string read_cases_fname;
   int f_read_cases_text;
   int read_cases_column_of_case;
   int read_cases_column_of_fname;
   int f_tasks;
   int nb_tasks;
   std::string tasks_line;

   create_file_description();
   ~create_file_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// file_io.cpp
// ##############################################################################

//! a collection of functions related to file io




class file_io {
public:
   file_io();
   ~file_io();

   void concatenate_files(
         std::string &fname_in_mask, int N,
         std::string &fname_out, std::string &EOF_marker,
         int f_title_line,
      int &cnt_total,
      std::vector<int> missing_idx,
      int verbose_level);
```

```
void concatenate_files_into(
      std::string &fname_in_mask, int N,
   std::ofstream &fp_out, std::string &EOF_marker,
   int f_title_line,
   int &cnt_total,
   std::vector<int> &missing_idx,
   int verbose_level);
void poset_classification_read_candidates_of_orbit(
      std::string &fname, int orbit_at_level,
   long int *&candidates, int &nb_candidates, int verbose_level);
void read_candidates_for_one_orbit_from_file(
      std::string &prefix,
      int level, int orbit_at_level, int level_of_candidates_file,
      long int *S,
      void (*early_test_func_callback)(long int *S, int len,
         long int *candidates, int nb_candidates,
         long int *good_candidates, int &nb_good_candidates,
         void *data, int verbose_level),
      void *early_test_func_callback_data,
      long int *&candidates,
      int &nb_candidates,
      int verbose_level);
int find_orbit_index_in_data_file(
      std::string &prefix,
      int level_of_candidates_file, long int *starter,
      int verbose_level);
void write_exact_cover_problem_to_file(int *Inc, int nb_rows,
   int nb_cols, std::string &fname);
void read_solution_file(
      std::string &fname,
   int *Inc, int nb_rows, int nb_cols,
   int *&Solutions, int &sol_length, int &nb_sol,
   int verbose_level);
void count_number_of_solutions_in_file_and_get_solution_size(
      std::string &fname,
   int &nb_solutions, int &solution_size,
   int verbose_level);
void count_number_of_solutions_in_file(
      std::string &fname,
   int &nb_solutions,
   int verbose_level);
void count_number_of_solutions_in_file_by_case(
      std::string &fname,
   int *&nb_solutions, int *&case_nb, int &nb_cases,
   int verbose_level);
void read_solutions_from_file_and_get_solution_size(
      std::string &fname,
```

```
   int &nb_solutions, long int *&Solutions, int &solution_size,
   int verbose_level);
void read_solutions_from_file(
      std::string &fname,
   int &nb_solutions, long int *&Solutions, int solution_size,
   int verbose_level);
void read_solutions_from_file_size_is_known(
      std::string &fname,
   std::vector<std::vector<long int> > &Solutions,
   int solution_size,
   int verbose_level);
void read_solutions_from_file_by_case(
      std::string &fname,
   int *nb_solutions, int *case_nb, int nb_cases,
   long int **&Solutions, int solution_size,
   int verbose_level);
void copy_file_to_ostream(
      std::ostream &ost, std::string &fname);
void int_vec_write_csv(
      int *v, int len,
      std::string &fname, std::string &label);
void lint_vec_write_csv(
      long int *v, int len,
      std::string &fname, std::string &label);
void int_vecs_write_csv(
      int *v1, int *v2, int len,
      std::string &fname, std::string &label1, std::string &label2);
void int_vecs3_write_csv(
      int *v1, int *v2, int *v3, int len,
      std::string &fname,
      std::string &label1, std::string &label2, std::string &label3);
void int_vec_array_write_csv(
      int nb_vecs, int **Vec, int len,
      std::string &fname, const char **column_label);
void lint_vec_array_write_csv(
      int nb_vecs, long int **Vec, int len,
      std::string &fname, const char **column_label);
void int_matrix_write_csv(
      std::string &fname, int *M, int m, int n);
void lint_matrix_write_csv(
      std::string &fname, long int *M, int m, int n);
void lint_matrix_write_csv_override_headers(
      std::string &fname,
      std::string *headers, long int *M, int m, int n);
void vector_write_csv(
      std::string &fname,
      std::vector<int > &V);
```

```
void vector_matrix_write_csv(
      std::string &fname,
      std::vector<std::vector<int> > &V);
void double_matrix_write_csv(
      std::string &fname,
   double *M, int m, int n);
void int_matrix_write_csv_with_labels(
      std::string &fname,
   int *M, int m, int n, const char **column_label);
void lint_matrix_write_csv_with_labels(
      std::string &fname,
   long int *M, int m, int n, const char **column_label);
void int_matrix_read_csv(
      std::string &fname, int *&M,
   int &m, int &n, int verbose_level);
void int_matrix_read_csv_no_border(
      std::string &fname,
   int *&M, int &m, int &n, int verbose_level);
void lint_matrix_read_csv_no_border(
      std::string &fname,
   long int *&M, int &m, int &n, int verbose_level);
void int_matrix_read_csv_data_column(
      std::string &fname,
   int *&M, int &m, int &n, int col_idx, int verbose_level);
void lint_matrix_read_csv_data_column(
      std::string &fname,
   long int *&M, int &m, int &n, int col_idx, int verbose_level);
void lint_matrix_read_csv(
      std::string &fname,
   long int *&M, int &m, int &n, int verbose_level);
void double_matrix_read_csv(
      std::string &fname, double *&M,
   int &m, int &n, int verbose_level);
void read_column_and_parse(
      std::string &fname, std::string &col_label,
      data_structures::set_of_sets *&SoS, int verbose_level);
void int_matrix_write_cas_friendly(
      std::string &fname, int *M, int m, int n);
void int_matrix_write_text(
      std::string &fname,
   int *M, int m, int n);
void lint_matrix_write_text(
      std::string &fname, long int *M, int m, int n);
void int_matrix_read_text(
      std::string &fname,
   int *&M, int &m, int &n);
void read_dimacs_graph_format(
```

```
      std::string &fname,
      int &nb_V, std::vector<std::vector<int> > &Edges,
      int verbose_level);
void parse_sets(
      int nb_cases, char **data, int f_casenumbers,
   int *&Set_sizes, long int **&Sets,
   char **&Ago_ascii, char **&Aut_ascii,
   int *&Casenumbers,
   int verbose_level);
void parse_sets_and_check_sizes_easy(
      int len, int nb_cases,
   char **data, long int **&sets);
void parse_line(
      char *line, int &len, long int *&set,
   char *ago_ascii, char *aut_ascii);
int count_number_of_orbits_in_file(
      std::string &fname, int verbose_level);
int count_number_of_lines_in_file(
      std::string &fname, int verbose_level);
int try_to_read_file(
      std::string &fname, int &nb_cases,
   char **&data, int verbose_level);
void read_and_parse_data_file(
      std::string &fname, int &nb_cases,
   char **&data, long int **&sets, int *&set_sizes, int verbose_level);
void free_data_fancy(
      int nb_cases,
   int *Set_sizes, long int **Sets,
   char **Ago_ascii, char **Aut_ascii,
   int *Casenumbers);
void read_and_parse_data_file_fancy(
      std::string &fname,
   int f_casenumbers,
   int &nb_cases,
   int *&Set_sizes, long int **&Sets,
   char **&Ago_ascii, char **&Aut_ascii,
   int *&Casenumbers,
   int verbose_level);
void read_set_from_file(
      std::string &fname,
   long int *&the_set, int &set_size, int verbose_level);
void write_set_to_file(
      std::string &fname,
   long int *the_set, int set_size, int verbose_level);
void read_set_from_file_lint(
      std::string &fname,
   long int *&the_set, int &set_size, int verbose_level);
```

```
void write_set_to_file_lint(
      std::string &fname,
   long int *the_set, int set_size, int verbose_level);
void read_set_from_file_int4(
      std::string &fname,
   long int *&the_set, int &set_size, int verbose_level);
void read_set_from_file_int8(
      std::string &fname,
   long int *&the_set, int &set_size, int verbose_level);
void write_set_to_file_as_int4(
      std::string &fname,
   long int *the_set, int set_size, int verbose_level);
void write_set_to_file_as_int8(
      std::string &fname,
   long int *the_set, int set_size, int verbose_level);
void read_k_th_set_from_file(
      std::string &fname, int k,
   int *&the_set, int &set_size, int verbose_level);
void write_incidence_matrix_to_file(
      std::string &fname,
   int *Inc, int m, int n, int verbose_level);
void read_incidence_matrix_from_inc_file(
      int *&M, int &m, int &n,
      std::string &inc_file_name, int inc_file_idx,
      int verbose_level);
void read_incidence_file(
      std::vector<std::vector<int> > &Geos,
      int &m, int &n, int &nb_flags,
      std::string &inc_file_name, int verbose_level);
void read_incidence_by_row_ranks_file(
      std::vector<std::vector<int> > &Geos,
      int &m, int &n, int &r,
      std::string &inc_file_name, int verbose_level);
int inc_file_get_number_of_geometries(
   char *inc_file_name, int verbose_level);
long int file_size(
      std::string &fname);
long int file_size(
      const char *name);
void delete_file(
      const char *fname);

void fwrite_int4(FILE *fp, int a);
int_4 fread_int4(FILE *fp);
void fwrite_uchars(FILE *fp, uchar *p, int len);
void fread_uchars(FILE *fp, uchar *p, int len);
```

```
void read_numbers_from_file(
      std::string &fname,
   int *&the_set, int &set_size, int verbose_level);
void read_ascii_set_of_sets_constant_size(
      std::string &fname_ascii,
      int *&Sets, int &nb_sets, int &set_size,
      int verbose_level);
void write_decomposition_stack(
      char *fname, int m, int n,
      int *v, int *b, int *aij, int verbose_level);
void create_file(
      create_file_description *Descr, int verbose_level);
void fix_escape_characters(char *str);
void create_files(
      create_file_description *Descr,
   int verbose_level);
void create_files_list_of_cases(
      data_structures::spreadsheet *S,
      create_file_description *Descr,
      int verbose_level);
int number_of_vertices_in_colored_graph(
      std::string &fname, int verbose_level);
void do_csv_file_select_rows(
      std::string &fname,
      std::string &rows_text,
      int verbose_level);
void do_csv_file_split_rows_modulo(
      std::string &fname,
      int split_modulo,
      int verbose_level);
void do_csv_file_select_cols(
      std::string &fname,
      std::string &cols_text,
      int verbose_level);
void do_csv_file_select_rows_and_cols(
      std::string &fname,
      std::string &rows_text, std::string &cols_text,
      int verbose_level);
void do_csv_file_extract_column_to_txt(
      std::string &csv_fname, std::string &col_label,
      int verbose_level);
void do_csv_file_sort_each_row(
      std::string &csv_fname, int verbose_level);
void do_csv_file_join(
      std::vector<std::string> &csv_file_join_fname,
      std::vector<std::string> &csv_file_join_identifier,
      int verbose_level);
```

```
void do_csv_file_concatenate(
      std::vector<std::string> &fname, std::string &fname_out,
      int verbose_level);
void do_csv_file_concatenate_from_mask(
      std::string &fname_in_mask, int N, std::string &fname_out,
      int verbose_level);
void do_csv_file_latex(std::string &fname,
      int f_produce_latex_header,
      int nb_lines_per_table,
      int verbose_level);
void read_solutions_and_tally(
      std::string &fname, int sz, int verbose_level);
void save_fibration(
      std::vector<std::vector<std::pair<int, int> > > &Fibration,
      std::string &fname, int verbose_level);
void save_cumulative_canonical_labeling(
      std::vector<std::vector<int> > &Cumulative_canonical_labeling,
      std::string &fname, int verbose_level);
void save_cumulative_ago(
      std::vector<long int> &Cumulative_Ago,
      std::string &fname, int verbose_level);
void save_cumulative_data(
      std::vector<std::vector<int> > &Cumulative_data,
      std::string &fname, int verbose_level);
void write_characteristic_matrix(
      std::string &fname,
      long int *data, int nb_rows, int data_sz, int nb_cols, int verbose_level);
void extract_from_makefile(
      std::string &fname,
      std::string &label,
      int f_tail, std::string &tail,
      std::vector<std::string> &text,
      int verbose_level);
void grade_statistic_from_csv(
      std::string &fname_csv,
      int f_midterm1, std::string &midterm1_label,
      int f_midterm2, std::string &midterm2_label,
      int f_final, std::string &final_label,
      int f_oracle_grade, std::string &oracle_grade_label,
      int verbose_level);
void count_solutions_in_list_of_files(
      int nb_files, std::string *fname,
      int *List_of_cases, int *&Nb_sol_per_file,
      int solution_size,
      int f_has_final_test_function,
      int (*final_test_function)(long int *data, int sz,
            void *final_test_data, int verbose_level),
```

```
         void *final_test_data,
         int verbose_level);
   void split_by_values(
         std::string &fname_in, int verbose_level);
   void change_values(
         std::string &fname_in, std::string &fname_out,
         std::string &input_values,
         std::string &output_values,
         int verbose_level);
   void read_file_as_array_of_strings(
         std::string &fname,
      std::string *&Lines,
      int &nb_lines,
      int verbose_level);

};


// ##############################################################################
// file_output.cpp
// ##############################################################################


//! a wrapper class for an ofstream which allows to store extra data

class file_output {
public:
   std::string fname;
   int f_file_is_open;
   std::ofstream *fp;
   void *user_data;

   file_output();
   ~file_output();
   void open(std::string &fname,
         void *user_data,
         int verbose_level);
   void close();
   void write_line(int nb, int *data,
         int verbose_level);
   void write_EOF(int nb_sol, int verbose_level);
};

// ##############################################################################
// latex_interface.cpp
// ##############################################################################
```

```
//! interface to create latex output files



class latex_interface {
public:
   latex_interface();
   ~latex_interface();
   void head_easy(std::ostream& ost);
   void head_easy_with_extras_in_the_praeamble(
         std::ostream& ost, std::string &extras);
   void head_easy_sideways(std::ostream& ost);
   void head(std::ostream& ost, int f_book, int f_title,
         std::string &title, std::string &author,
      int f_toc, int f_landscape, int f_12pt,
      int f_enlarged_page, int f_pagenumbers,
      std::string &extras_for_preamble);
   void foot(std::ostream& ost);

   // two functions from DISCRETA1:

   void incma_latex_with_text_labels(
         std::ostream &fp,
         graphics::draw_incidence_structure_description *Descr,
      int v, int b,
      int V, int B, int *Vi, int *Bj,
      int *incma,
      int f_labelling_points, std::string *point_labels,
      int f_labelling_blocks, std::string *block_labels,
      int verbose_level);
   void incma_latex(std::ostream &fp,
      int v, int b,
      int V, int B, int *Vi, int *Bj,
      int *incma,
      int verbose_level);
   void incma_latex_with_labels(std::ostream &fp,
      int v, int b,
      int V, int B, int *Vi, int *Bj,
      int *row_labels_int,
      int *col_labels_int,
      int *incma,
      int verbose_level);
   void print_01_matrix_tex(
         std::ostream &ost, int *p, int m, int n);
   void print_integer_matrix_tex(
         std::ostream &ost, int *p, int m, int n);
```

```
void print_lint_matrix_tex(
      std::ostream &ost,
   long int *p, int m, int n);
void print_longinteger_matrix_tex(
      std::ostream &ost,
      ring_theory::longinteger_object *p, int m, int n);
void print_integer_matrix_with_labels(
      std::ostream &ost, int *p,
   int m, int n, int *row_labels, int *col_labels, int f_tex);
void print_lint_matrix_with_labels(
      std::ostream &ost,
   long int *p, int m, int n,
   long int *row_labels, long int *col_labels,
   int f_tex);
void print_integer_matrix_with_standard_labels(
      std::ostream &ost,
   int *p, int m, int n, int f_tex);
void print_lint_matrix_with_standard_labels(
      std::ostream &ost,
   long int *p, int m, int n, int f_tex);
void print_integer_matrix_with_standard_labels_and_offset(
      std::ostream &ost,
   int *p, int m, int n, int m_offset, int n_offset, int f_tex);
void print_lint_matrix_with_standard_labels_and_offset(
      std::ostream &ost,
   long int *p, int m, int n, int m_offset, int n_offset, int f_tex);
void print_integer_matrix_tex_block_by_block(
      std::ostream &ost,
   int *p, int m, int n, int block_width);
void print_integer_matrix_with_standard_labels_and_offset_text(
      std::ostream &ost,
   int *p, int m, int n, int m_offset, int n_offset);
void print_lint_matrix_with_standard_labels_and_offset_text(
   std::ostream &ost, long int *p, int m, int n,
   int m_offset, int n_offset);
void print_integer_matrix_with_standard_labels_and_offset_tex(
      std::ostream &ost,
   int *p, int m, int n, int m_offset, int n_offset);
void print_lint_matrix_with_standard_labels_and_offset_tex(
   std::ostream &ost, long int *p, int m, int n,
   int m_offset, int n_offset);
void print_big_integer_matrix_tex(
      std::ostream &ost, int *p, int m, int n);
void int_vec_print_as_matrix(std::ostream &ost,
   int *v, int len, int width, int f_tex);
void lint_vec_print_as_matrix(std::ostream &ost,
   long int *v, int len, int width, int f_tex);
```

```
void int_matrix_print_with_labels_and_partition(
      std::ostream &ost,
   int *p, int m, int n,
   int *row_labels, int *col_labels,
   int *row_part_first, int *row_part_len, int nb_row_parts,
   int *col_part_first, int *col_part_len, int nb_col_parts,
   void (*process_function_or_NULL)(int *p, int m, int n,
      int i, int j, int val, std::string &output, void *data),
   void *data,
   int f_tex);
void lint_matrix_print_with_labels_and_partition(
      std::ostream &ost,
   long int *p, int m, int n,
   int *row_labels, int *col_labels,
   int *row_part_first, int *row_part_len, int nb_row_parts,
   int *col_part_first, int *col_part_len, int nb_col_parts,
   void (*process_function_or_NULL)(long int *p, int m, int n,
      int i, int j, int val, std::string &output, void *data),
   void *data,
   int f_tex);
void int_matrix_print_tex(
      std::ostream &ost, int *p, int m, int n);
void lint_matrix_print_tex(
      std::ostream &ost, long int *p, int m, int n);
void print_cycle_tex_with_special_point_labels(
      std::ostream &ost, int *pts, int nb_pts,
      void (*point_label)(std::stringstream &sstr,
            int pt, void *data),
      void *point_label_data);
void int_set_print_tex(
      std::ostream &ost, int *v, int len);
void lint_set_print_tex(
      std::ostream &ost, long int *v, int len);
void lint_set_print_tex_text_mode(
      std::ostream &ost, long int *v, int len);
void print_type_vector_tex(
      std::ostream &ost, int *v, int len);
void int_set_print_masked_tex(
      std::ostream &ost,
   int *v, int len, const char *mask_begin, const char *mask_end);
void lint_set_print_masked_tex(
      std::ostream &ost,
   long int *v, int len,
   const char *mask_begin,
   const char *mask_end);
void int_set_print_tex_for_inline_text(
      std::ostream &ost,
```

```
         int *v, int len);
   void lint_set_print_tex_for_inline_text(
         std::ostream &ost,
         long int *v, int len);
   void latexable_string(
         std::stringstream &str,
         const char *p, int max_len, int line_skip);
   void print_row_tactical_decomposition_scheme_tex(
      std::ostream &ost, int f_enter_math_mode,
      long int *row_class_size, int nb_row_classes,
      long int *col_class_size, int nb_col_classes,
      long int *row_scheme);
   void print_column_tactical_decomposition_scheme_tex(
      std::ostream &ost, int f_enter_math_mode,
      long int *row_class_size, int nb_row_classes,
      long int *col_class_size, int nb_col_classes,
      long int *col_scheme);

};



// ##############################################################################
// mem_object_registry_entry.cpp
// ##############################################################################

//! a class related to mem_object_registry


class mem_object_registry_entry {
public:
   int time_stamp;
   void *pointer;
   int object_type;
   long int object_n;
   int object_size_of;
      // needed for objects of type class
   const char *extra_type_info;
   const char *source_file;
   int source_line;

   mem_object_registry_entry();
   ~mem_object_registry_entry();
   void null();
   void set_type_from_string(char *str);
   void print_type(std::ostream &ost);
   int size_of();
```

```
   void print(int line);
   void print_csv(std::ostream &ost, int line);
};


// ##############################################################################
// mem_object_registry.cpp:
// ##############################################################################




#define REGISTRY_SIZE 1000
#define POINTER_TYPE_INVALID 0
#define POINTER_TYPE_int 1
#define POINTER_TYPE_pint 2
#define POINTER_TYPE_lint 3
#define POINTER_TYPE_plint 4
#define POINTER_TYPE_ppint 5
#define POINTER_TYPE_pplint 6
#define POINTER_TYPE_char 7
#define POINTER_TYPE_uchar 8
#define POINTER_TYPE_pchar 9
#define POINTER_TYPE_puchar 10
#define POINTER_TYPE_PVOID 11
#define POINTER_TYPE_OBJECT 12
#define POINTER_TYPE_OBJECTS 13



//! maintains a registry of allocated memory



class mem_object_registry {
public:
   int f_automatic_dump;
   int automatic_dump_interval;
   char automatic_dump_fname_mask[1000];

   int nb_entries_allocated;
   int nb_entries_used;
   mem_object_registry_entry *entries;
      // entries are sorted by
      // the value of the pointer

   int nb_allocate_total;
```

```
   // total number of allocations
int nb_delete_total;
   // total number of deletions
int cur_time;
   // increments with every allocate and every delete

int f_ignore_duplicates;
   // do not complain about duplicate entries
int f_accumulate;
   // do not remove entries when deleting memory

mem_object_registry();
~mem_object_registry();
void init(int verbose_level);
void accumulate_and_ignore_duplicates(int verbose_level);
void allocate(int N, int verbose_level);
void set_automatic_dump(
      int automatic_dump_interval, const char *fname_mask,
      int verbose_level);
void automatic_dump();
void manual_dump();
void manual_dump_with_file_name(const char *fname);
void dump();
void dump_to_csv_file(const char *fname);
int *allocate_int(long int n, const char *file, int line);
void free_int(int *p, const char *file, int line);
int **allocate_pint(long int n, const char *file, int line);
void free_pint(int **p, const char *file, int line);
long int *allocate_lint(long int n, const char *file, int line);
void free_lint(long int *p, const char *file, int line);
long int **allocate_plint(long int n, const char *file, int line);
void free_plint(long int **p, const char *file, int line);
int ***allocate_ppint(long int n, const char *file, int line);
void free_ppint(int ***p, const char *file, int line);
long int ***allocate_pplint(long int n, const char *file, int line);
void free_pplint(long int ***p, const char *file, int line);
char *allocate_char(long int n, const char *file, int line);
void free_char(char *p, const char *file, int line);
uchar *allocate_uchar(long int n, const char *file, int line);
void free_uchar(uchar *p, const char *file, int line);
char **allocate_pchar(long int n, const char *file, int line);
void free_pchar(char **p, const char *file, int line);
uchar **allocate_puchar(long int n, const char *file, int line);
void free_puchar(uchar **p, const char *file, int line);
void **allocate_pvoid(long int n, const char *file, int line);
void free_pvoid(void **p, const char *file, int line);
void *allocate_OBJECTS(void *p, long int n, std::size_t size_of,
```

```
         const char *extra_type_info, const char *file, int line);
   void free_OBJECTS(void *p, const char *file, int line);
   void *allocate_OBJECT(void *p, std::size_t size_of,
         const char *extra_type_info, const char *file, int line);
   void free_OBJECT(void *p, const char *file, int line);
   int search(void *p, int &idx);
   void insert_at(int idx);
   void add_to_registry(void *pointer,
         int object_type, long int object_n, int object_size_of,
         const char *extra_type_info,
         const char *source_file, int source_line,
         int verbose_level);
   void delete_from_registry(void *pointer, int verbose_level);
   void sort_by_size(int verbose_level);
   void sort_by_location_and_get_frequency(int verbose_level);
   void sort_by_type(int verbose_level);
   void sort_by_location(int verbose_level);
};



// ##############################################################################
// memory_object.cpp
// ##############################################################################

//! for serialization of complex data types




class memory_object {
public:
   memory_object();
   ~memory_object();

   char *data;
   long int alloc_length;
      // maintained by alloc()
   long int used_length;
   long int cur_pointer;


   char & s_i(int i) { return data[i]; };
   void init(long int length, char *initial_data, int verbose_level);
   void alloc(long int length, int verbose_level);
   void append(long int length, char *d, int verbose_level);
   void realloc(long int &new_length, int verbose_level);
```

```
   void write_char(char c);
   void read_char(char *c);
   void write_string(const char *p);
   void write_string(std::string &p);
   void read_string(std::string &p);
   void write_double(double f);
   void read_double(double *f);
   void write_lint(long int i);
   void read_lint(long int *i);
   void write_int(int i);
   void read_int(int *i);
   void read_file(std::string &fname, int verbose_level);
   void write_file(std::string &fname, int verbose_level);
   int multiplicity_of_character(char c);
};


// ##############################################################################
// numerics.cpp
// ##############################################################################



//! numerical functions, mostly concerned with double

class numerics {
public:
   numerics();
   ~numerics();
   void vec_print(double *a, int len);
   void vec_linear_combination1(
         double c1, double *v1,
         double *w, int len);
   void vec_linear_combination(
         double c1, double *v1,
         double c2, double *v2, double *v3, int len);
   void vec_linear_combination3(
         double c1, double *v1,
         double c2, double *v2,
         double c3, double *v3,
         double *w, int len);
   void vec_add(
         double *a, double *b, double *c, int len);
   void vec_subtract(
         double *a, double *b, double *c, int len);
   void vec_scalar_multiple(
         double *a, double lambda, int len);
```

```
int Gauss_elimination(
      double *A, int m, int n,
   int *base_cols, int f_complete,
   int verbose_level);
void print_system(double *A, int m, int n);
void get_kernel(double *M, int m, int n,
   int *base_cols, int nb_base_cols,
   int &kernel_m, int &kernel_n,
   double *kernel);
// kernel must point to the appropriate amount of memory!
// (at least n * (n - nb_base_cols) doubles)
// m is not used!
int Null_space(double *M, int m, int n, double *K,
   int verbose_level);
// K will be k x n
// where k is the return value.
void vec_normalize_from_back(double *v, int len);
void vec_normalize_to_minus_one_from_back(
      double *v, int len);
int triangular_prism(
      double *P1, double *P2, double *P3,
   double *abc3, double *angles3, double *T3,
   int verbose_level);
int general_prism(
      double *Pts, int nb_pts, double *Pts_xy,
   double *abc3, double *angles3, double *T3,
   int verbose_level);
void mult_matrix(
      double *v, double *R, double *vR);
void mult_matrix_matrix(
      double *A, double *B, double *C,
      int m, int n, int o);
// A is m x n, B is n x o, C is m x o
void print_matrix(double *R);
void make_Rz(double *R, double phi);
void make_Ry(double *R, double psi);
void make_Rx(double *R, double chi);
double atan_xy(double x, double y);
double dot_product(
      double *u, double *v, int len);
void cross_product(
      double *u, double *v, double *n);
double distance_euclidean(
      double *x, double *y, int len);
double distance_from_origin(
      double x1, double x2, double x3);
double distance_from_origin(double *x, int len);
```

```
void make_unit_vector(double *v, int len);
void center_of_mass(
      double *Pts, int len,
      int *Pt_idx, int nb_pts, double *c);
void plane_through_three_points(
      double *p1, double *p2, double *p3,
   double *n, double &d);
void orthogonal_transformation_from_point_to_basis_vector(
   double *from,
   double *A, double *Av, int verbose_level);
void output_double(
      double a, std::ostream &ost);
void mult_matrix_4x4(
      double *v, double *R, double *vR);
void transpose_matrix_4x4(
      double *A, double *At);
void transpose_matrix_nxn(
      double *A, double *At, int n);
void substitute_quadric_linear(
      double *coeff_in, double *coeff_out,
   double *A4_inv, int verbose_level);
void substitute_cubic_linear_using_povray_ordering(
      double *coeff_in, double *coeff_out,
      double *A4_inv, int verbose_level);
void substitute_quartic_linear_using_povray_ordering(
   double *coeff_in, double *coeff_out,
   double *A4_inv, int verbose_level);
void make_transform_t_varphi_u_double(
      int n, double *varphi, double *u,
   double *A, double *Av, int verbose_level);
// varphi are the dual coordinates of a plane.
// u is a vector such that varphi(u) \neq -1.
// A = I + varphi * u.
void matrix_double_inverse(
      double *A, double *Av, int n,
      int verbose_level);
int line_centered(
      double *pt1_in, double *pt2_in,
   double *pt1_out, double *pt2_out, double r,
   int verbose_level);
int line_centered_tolerant(
      double *pt1_in, double *pt2_in,
   double *pt1_out, double *pt2_out, double r,
   int verbose_level);
int sign_of(double a);
void eigenvalues(
      double *A, int n, double *lambda,
```

```
      int verbose_level);
void eigenvectors(double *A, double *Basis,
      int n, double *lambda, int verbose_level);
double rad2deg(double phi);
void vec_copy(double *from, double *to, int len);
void vec_swap(double *from, double *to, int len);
void vec_print(std::ostream &ost, double *v, int len);
void vec_scan(const char *s, double *&v, int &len);
void vec_scan(std::string &s, double *&v, int &len);
void vec_scan_from_stream(
      std::istream & is, double *&v, int &len);


double cos_grad(double phi);
double sin_grad(double phi);
double tan_grad(double phi);
double atan_grad(double x);
void adjust_coordinates_double(
      double *Px, double *Py, int *Qx, int *Qy,
   int N,
   double xmin, double ymin, double xmax, double ymax,
   int verbose_level);
void Intersection_of_lines(
      double *X, double *Y,
   double *a, double *b, double *c,
   int l1, int l2, int pt);
void intersection_of_lines(
      double a1, double b1, double c1,
   double a2, double b2, double c2,
   double &x, double &y);
void Line_through_points(
      double *X, double *Y,
   double *a, double *b, double *c,
   int pt1, int pt2, int line_idx);
void line_through_points(double pt1_x, double pt1_y,
   double pt2_x, double pt2_y,
   double &a, double &b, double &c);
void intersect_circle_line_through(
      double rad, double x0, double y0,
   double pt1_x, double pt1_y,
   double pt2_x, double pt2_y,
   double &x1, double &y1, double &x2, double &y2);
void intersect_circle_line(
      double rad, double x0, double y0,
   double a, double b, double c,
   double &x1, double &y1, double &x2, double &y2);
void affine_combination(double *X, double *Y,
```

```
      int pt0, int pt1, int pt2, double alpha, int new_pt);
   void on_circle_double(double *Px, double *Py, int idx,
      double angle_in_degree, double rad);
   void affine_pt1(int *Px, int *Py, int p0, int p1, int p2,
      double f1, int p3);
   void affine_pt2(int *Px, int *Py, int p0, int p1, int p1b,
      double f1, int p2, int p2b, double f2, int p3);
   double norm_of_vector_2D(int x1, int x2, int y1, int y2);
   void transform_llur(int *in, int *out, int &x, int &y);
   void transform_dist(int *in, int *out, int &x, int &y);
   void transform_dist_x(int *in, int *out, int &x);
   void transform_dist_y(int *in, int *out, int &y);
   void transform_llur_double(
         double *in, double *out, double &x, double &y);
   void on_circle_int(
         int *Px, int *Py, int idx, int angle_in_degree, int rad);
   double power_of(double x, int n);
   double bernoulli(double p, int n, int k);
   void local_coordinates_wrt_triangle(double *pt,
         double *triangle_points, double &x, double &y,
         int verbose_level);
   int intersect_line_and_line(
         double *line1_pt1_coords, double *line1_pt2_coords,
         double *line2_pt1_coords, double *line2_pt2_coords,
         double &lambda,
         double *pt_coords,
         int verbose_level);
   void clebsch_map_up(
         double *line1_pt1_coords, double *line1_pt2_coords,
         double *line2_pt1_coords, double *line2_pt2_coords,
      double *pt_in, double *pt_out,
      double *Cubic_coords_povray_ordering,
      int line1_idx, int line2_idx,
      int verbose_level);
   void project_to_disc(
         int f_projection_on, int f_transition,
         int step, int nb_steps,
      double rad, double height, double x, double y,
      double &xp, double &yp);

};



// ##############################################################################
// orbiter_data_file.cpp
// ##############################################################################
```

```
//! read output files from the poset classification



class orbiter_data_file {
public:
   int nb_cases;
   long int **sets;
   int *set_sizes;
   char **Ago_ascii;
   char **Aut_ascii;
   int *Casenumbers;

   orbiter_data_file();
   ~orbiter_data_file();
   void load(std::string &fname, int verbose_level);
};


// ##############################################################################
// orbiter_session.cpp
// ##############################################################################


//! global Orbiter session
extern orbiter_kernel_system::orbiter_session *Orbiter;


//! The orbiter session is responsible for the command line interface and the pro
gram execution. There will only be one instance of this object


class orbiter_session {

public:

   int verbose_level;

   int t0;

   int f_draw_options;
   graphics::layered_graph_draw_options *draw_options;
```

```
int f_draw_incidence_structure_description;
graphics::draw_incidence_structure_description
   *Draw_incidence_structure_description;

int f_list_arguments;

int f_seed;
int the_seed;

int f_memory_debug;
int memory_debug_verbose_level;

int f_override_polynomial;
std::string override_polynomial;

int f_orbiter_path;
std::string orbiter_path;

int f_magma_path;
std::string magma_path;

int f_fork;
int fork_argument_idx;
std::string fork_variable;
std::string fork_logfile_mask;
int fork_from;
int fork_to;
int fork_step;

orbiter_kernel_system::orbiter_symbol_table *Orbiter_symbol_table;

long int nb_times_finite_field_created;
long int nb_times_projective_space_created;
long int nb_times_action_created;
long int nb_calls_to_densenauty;

data_structures::int_vec *Int_vec;
data_structures::lint_vec *Lint_vec;

orbiter_kernel_system::mem_object_registry *global_mem_object_registry;

int longinteger_f_print_scientific;
int syntax_tree_node_index;

orbiter_session();
~orbiter_session();
void print_help(int argc,
```

```
        std::string *argv, int i, int verbose_level);
  int recognize_keyword(int argc,
        std::string *argv, int i, int verbose_level);
  int read_arguments(int argc,
        std::string *argv, int i0);
  void fork(
        int argc, std::string *argv, int verbose_level);

  void *get_object(int idx);
  symbol_table_object_type get_object_type(int idx);
  int find_symbol(std::string &label);
  void get_vector_from_label(
        std::string &label, long int *&v, int &sz, int verbose_level);
  void get_int_vector_from_label(
        std::string &label, int *&v, int &sz, int verbose_level);
  void get_lint_vector_from_label(
        std::string &label, long int *&v, int &sz, int verbose_level);
  void get_matrix_from_label(
        std::string &label,
        int *&v, int &m, int &n);
  void find_symbols(
        std::vector<std::string> &Labels, int *&Idx);
  void print_symbol_table();
  void add_symbol_table_entry(
        std::string &label,
        orbiter_symbol_table_entry *Symb, int verbose_level);
  void get_lint_vec(
        std::string &label,
        long int *&the_set, int &set_size, int verbose_level);
  void print_type(
        symbol_table_object_type t);
  field_theory::finite_field
        *get_object_of_type_finite_field(
        std::string &label);
  ring_theory::homogeneous_polynomial_domain
        *get_object_of_type_polynomial_ring(
        std::string &label);
  data_structures::vector_builder *get_object_of_type_vector(
        std::string &label);
  void start_memory_debug();
  void stop_memory_debug();

};


// ##############################################################################
// orbiter_symbol_table_entry.cpp
```

```
// ##############################################################################


enum symbol_table_entry_type {
   t_nothing,
   t_intvec,
   t_object,
   t_string,
};



//! symbol table to store data entries for the orbiter run-time system


class orbiter_symbol_table_entry {
public:
   std::string label;
   enum symbol_table_entry_type type;
   enum symbol_table_object_type object_type;
   int *vec;
   int vec_len;
   std::string str;
   void *ptr;

   orbiter_symbol_table_entry();
   ~orbiter_symbol_table_entry();
   void freeself();
   void init(std::string &str_label);
   void init_finite_field(std::string &label,
         field_theory::finite_field *F, int verbose_level);
   void init_polynomial_ring(std::string &label,
         ring_theory::homogeneous_polynomial_domain *HPD, int verbose_level);
   void init_any_group(std::string &label,
         void *p, int verbose_level);
   void init_linear_group(std::string &label,
         void *p, int verbose_level);
   void init_permutation_group(std::string &label,
         void *p, int verbose_level);
   void init_modified_group(std::string &label,
         void *p, int verbose_level);
   void init_projective_space(std::string &label,
         void *p, int verbose_level);
   void init_orthogonal_space(std::string &label,
         void *p, int verbose_level);
   void init_BLT_set_classify(std::string &label,
         void *p, int verbose_level);
```

```
void init_spread_classify(std::string &label,
      void *p, int verbose_level);
void init_formula(std::string &label,
      void *p, int verbose_level);
void init_cubic_surface(std::string &label,
      void *p, int verbose_level);
void init_quartic_curve(std::string &label,
      void *p, int verbose_level);
void init_BLT_set(std::string &label,
      void *p, int verbose_level);
void init_classification_of_cubic_surfaces_with_double_sixes(
      std::string &label,
      void *p, int verbose_level);
void init_collection(std::string &label,
      std::string &list_of_objects, int verbose_level);
void init_geometric_object(std::string &label,
      geometry::geometric_object_create *COC, int verbose_level);
void init_graph(std::string &label,
      void *Gr, int verbose_level);
void init_code(std::string &label,
      void *Code, int verbose_level);
void init_spread(std::string &label,
      void *Spread, int verbose_level);
void init_translation_plane(std::string &label,
      void *Tp, int verbose_level);
void init_spread_table(std::string &label,
      void *P, int verbose_level);
void init_packing_was(std::string &label,
      void *P, int verbose_level);
void init_packing_was_choose_fixed_points(std::string &label,
      void *P, int verbose_level);
void init_packing_long_orbits(std::string &label,
      void *PL, int verbose_level);
void init_graph_classify(std::string &label,
      void *GC, int verbose_level);
void init_diophant(std::string &label,
      void *Dio, int verbose_level);
void init_design(std::string &label,
      void *DC, int verbose_level);
void init_design_table(std::string &label,
      void *DT, int verbose_level);
void init_large_set_was(std::string &label,
      void *LSW, int verbose_level);
void init_set(std::string &label,
      void *SB, int verbose_level);
void init_vector(std::string &label,
      void *VB, int verbose_level);
```

```
   void init_combinatorial_objects(std::string &label,
         data_structures::data_input_stream *IS, int verbose_level);
   void init_geometry_builder_object(std::string &label,
         geometry_builder::geometry_builder *GB, int verbose_level);
   void init_vector_ge(std::string &label,
         void *V, int verbose_level);
   void init_action_on_forms(std::string &label,
         void *AF, int verbose_level);
   void init_orbits(std::string &label,
         void *OC, int verbose_level);
   void init_poset_classification_control(std::string &label,
         void *PCC, int verbose_level);
   void print();
};


// ##############################################################################
// orbiter_symbol_table.cpp
// ##############################################################################




//! symbol table to store data entries for the orbiter run-time system


class orbiter_symbol_table {
public:
   std::vector<orbiter_symbol_table_entry> Table;

   orbiter_symbol_table();
   ~orbiter_symbol_table();
   int find_symbol(std::string &str);
   void add_symbol_table_entry(std::string &str,
         orbiter_symbol_table_entry *Symb, int verbose_level);
   void print_symbol_table();
   void *get_object(int idx);
   symbol_table_object_type get_object_type(int idx);
   void print_type(symbol_table_object_type t);

};




// ##############################################################################
// os_interface.cpp
```

```
// ##############################################################################


//! interface to system functions

class os_interface {
public:

   void runtime(long *l);
   int os_memory_usage();
   int os_ticks();
   int os_ticks_system();
   int os_ticks_per_second();
   void os_ticks_to_dhms(int ticks, int tps,
         int &d, int &h, int &m, int &s);
   void time_check_delta(std::ostream &ost, int dt);
   void print_elapsed_time(std::ostream &ost, int d, int h, int m, int s);
   void time_check(std::ostream &ost, int t0);
   int delta_time(int t0);
   void seed_random_generator_with_system_time();
   void seed_random_generator(int seed);
   int random_integer(int p);
   int os_seconds_past_1970();
   void get_string_from_command_line(std::string &p, int argc, std::string *argv,
         int &i, int verbose_level);
   void test_swap();
   void block_swap_chars(char *ptr, int size, int no);
   void code_int4(char *&p, int_4 i);
   int_4 decode_int4(const char *&p);
   void code_uchar(char *&p, unsigned char a);
   void decode_uchar(const char *&p, unsigned char &a);
   void get_date(std::string &str);
   void test_typedefs();

};


// ##############################################################################
// override_double.cpp
// ##############################################################################


//! to temporarily override a double variable with a new value

class override_double {
public:
   double *p;
```

```
   double original_value;
   double new_value;

   override_double(double *p, double value);
   ~override_double();
};


// ##############################################################################
// prepare_frames.cpp
// ##############################################################################


//! to prepare files using a unified file naming scheme

class prepare_frames {
public:

   int nb_inputs;
   int input_first[1000];
   int input_len[1000];
   std::string input_mask[1000];
   int f_o;
   std::string output_mask;
   int f_output_starts_at;
   int output_starts_at;
   int f_step;
   int step;

   prepare_frames();
   ~prepare_frames();
   int parse_arguments(int argc, std::string *argv, int verbose_level);
   void print();
   void print_item(int i);
   void do_the_work(int verbose_level);
};




}}}




#endif /* ORBITER_SRC_LIB_FOUNDATIONS_IO_AND_OS_IO_AND_OS_H_ */
```

3.19 Orthogonal Geometry

```
/*
 * orthogonal.h
 *
 *  Created on: Dec 8, 2020
 *      Author: betten
 */

#ifndef SRC_LIB_FOUNDATIONS_ORTHOGONAL_ORTHOGONA_H_
#define SRC_LIB_FOUNDATIONS_ORTHOGONAL_ORTHOGONA_H_

namespace orbiter {
namespace layer1_foundations {
namespace orthogonal_geometry {


// ##############################################################################
// blt_set_domain.cpp
// ##############################################################################

//! BLT-sets in Q(4,q)



class blt_set_domain {

public:
   field_theory::finite_field *F;
   int f_semilinear; // from the command line
   int epsilon; // the type of the quadric (0, 1 or -1)
   int n; // algebraic dimension
   int q; // field order
   int target_size; // q + 1
   int degree; // number of points on the quadric

   std::string prefix; // "BLT_q%d"

   orthogonal *O;
   int f_orthogonal_allocated;


   int *Pts; // [target_size * n]
   int *Candidates; // [degree * n]

   geometry::projective_space *P;
   geometry::grassmann *G53;
   geometry::grassmann *G54;
```

```
   geometry::grassmann *G43;


   blt_set_domain();
   ~blt_set_domain();
   void init(orthogonal *O,
      int verbose_level);
   long int intersection_of_hyperplanes(
         long int plane_rk1, long int plane_rk2,
         int verbose_level);
   long int compute_tangent_hyperplane(
      long int pt,
      int verbose_level);
   void report_given_point_set(std::ostream &ost,
         long int *Pts, int nb_pts, int verbose_level);
   void compute_adjacency_list_fast(int first_point_of_starter,
      long int *points, int nb_points, int *point_color,
      data_structures::bitvector *&Bitvec,
      int verbose_level);
   void compute_colors(int orbit_at_level,
      long int *starter, int starter_sz,
      long int special_line,
      long int *candidates, int nb_candidates,
      int *&point_color, int &nb_colors,
      int verbose_level);
   void early_test_func(long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   int pair_test(int a, int x, int y, int verbose_level);
   int check_conditions(int len, long int *S, int verbose_level);
   int collinearity_test(long int *S, int len, int verbose_level);
   void print(std::ostream &ost, long int *S, int len);
   void find_free_points(long int *S, int S_sz,
         long int *&free_pts, int *&free_pt_idx, int &nb_free_pts,
      int verbose_level);
   int create_graph(
      int case_number, int nb_cases_total,
      long int *Starter_set, int starter_size,
      long int *candidates, int nb_candidates,
      int f_eliminate_graphs_if_possible,
      graph_theory::colored_graph *&CG,
      int verbose_level);
};


// ##############################################################################
```

```
// blt_set_invariants.cpp
// ##############################################################################

//! invariants of a BLT-sets in Q(4,q)



class blt_set_invariants {

public:

   blt_set_domain *D;

   int set_size; // = D->q + 1
   long int *the_set_in_orthogonal; // [set_size]
   long int *the_set_in_PG; // [set_size]

   int *intersection_type;
   int highest_intersection_number;
   int *intersection_matrix;
   int nb_planes;

   data_structures::set_of_sets *Sos;
   data_structures::set_of_sets *Sos2;
   data_structures::set_of_sets *Sos3;

   geometry::decomposition *D2;
   geometry::decomposition *D3;

   int *Sos2_idx;
   int *Sos3_idx;

   blt_set_invariants();
   ~blt_set_invariants();
   void init(blt_set_domain *D, long int *the_set,
      int verbose_level);
   void compute(int verbose_level);
   void latex(std::ostream &ost, int verbose_level);
};



// ##############################################################################
// hyperbolic_pair.cpp
// ##############################################################################

//! tactical decomposition of O^epsilon(n,q) according to a hyperbolic pair
```

```
class hyperbolic_pair {

public:

   orthogonal *O;

   field_theory::finite_field *F;

   int q;
   int epsilon;
   int m;
   int n;

   long int pt_P, pt_Q;
   long int nb_points;
   long int nb_lines;

   long int T1_m;
   long int T1_mm1;
   long int T1_mm2;
   long int T2_m;
   long int T2_mm1;
   long int T2_mm2;
   long int N1_m;
   long int N1_mm1;
   long int N1_mm2;
   long int S_m;
   long int S_mm1;
   long int S_mm2;
   long int Sbar_m;
   long int Sbar_mm1;
   long int Sbar_mm2;

   long int alpha; // number of points in the subspace
   long int beta; // number of points in the subspace of the subspace
   long int gamma; // = alpha * beta / (q + 1);
   int subspace_point_type;
   int subspace_line_type;

   int nb_point_classes, nb_line_classes;
   long int *A, *B, *P, *L;

   // for hyperbolic:
   long int p1, p2, p3, p4, p5, p6;
   long int l1, l2, l3, l4, l5, l6, l7;
```

```
long int a11, a12, a22, a23, a26, a32, a34, a37;
long int a41, a43, a44, a45, a46, a47, a56, a67;
long int b11, b12, b22, b23, b26, b32, b34, b37;
long int b41, b43, b44, b45, b46, b47, b56, b67;



// additionally, for parabolic:
long int p7, l8;
long int a21, a36, a57, a22a, a33, a22b;
long int a32b, a42b, a51, a53, a54, a55, a66, a77;
long int b21, b36, b57, b22a, b33, b22b;
long int b32b, b42b, b51, b53, b54, b55, b66, b77;
long int a12b, a52a;
long int b12b, b52a;
long int delta, omega, lambda, mu, nu, zeta;
// parabolic q odd requires square / non-square tables


int *v1, *v2, *v3, *v4, *v5, *v_tmp;
int *v_tmp2; // for use in parabolic_type_and_index_to_point_rk
int *v_neighbor5;


// stuff for rank_point
int *rk_pt_v;


hyperbolic_pair();
~hyperbolic_pair();
void init(orthogonal *O, int verbose_level);
void init_counting_functions(int verbose_level);
void init_decomposition(int verbose_level);
void init_parabolic(int verbose_level);
void init_parabolic_even(int verbose_level);
void init_parabolic_odd(int verbose_level);
void init_hyperbolic(int verbose_level);
void fill(long int *M, int i, int j, long int a);
void print_schemes();


// hyperbolic_pair_hyperbolic.cpp:
long int hyperbolic_type_and_index_to_point_rk(
      long int type,
      long int index, int verbose_level);
void hyperbolic_point_rk_to_type_and_index(
      long int rk,
```

```
      long int &type, long int &index);

void hyperbolic_unrank_line(long int &p1, long int &p2,
   long int rk, int verbose_level);
long int hyperbolic_rank_line(long int p1, long int p2,
      int verbose_level);

void unrank_line_L1(long int &p1, long int &p2,
      long int index, int verbose_level);
long int rank_line_L1(long int p1, long int p2,
      int verbose_level);
void unrank_line_L2(long int &p1, long int &p2,
      long int index, int verbose_level);
long int rank_line_L2(long int p1, long int p2,
      int verbose_level);
void unrank_line_L3(long int &p1, long int &p2,
      long int index, int verbose_level);
long int rank_line_L3(long int p1, long int p2,
      int verbose_level);
void unrank_line_L4(long int &p1, long int &p2,
      long int index, int verbose_level);
long int rank_line_L4(long int p1, long int p2,
      int verbose_level);
void unrank_line_L5(long int &p1, long int &p2,
      long int index, int verbose_level);
long int rank_line_L5(long int p1, long int p2,
      int verbose_level);
void unrank_line_L6(long int &p1, long int &p2,
      long int index, int verbose_level);
long int rank_line_L6(long int p1, long int p2,
      int verbose_level);
void unrank_line_L7(long int &p1, long int &p2,
      long int index, int verbose_level);
long int rank_line_L7(long int p1, long int p2,
      int verbose_level);

void hyperbolic_canonical_points_of_line(
      int line_type,
      long int pt1, long int pt2,
      long int &cpt1, long int &cpt2,
   int verbose_level);

void canonical_points_L1(long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
void canonical_points_L2(long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
void canonical_points_L3(long int pt1, long int pt2,
```

```
      long int &cpt1, long int &cpt2);
void canonical_points_L4(long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
void canonical_points_L5(long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
void canonical_points_L6(long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
void canonical_points_L7(long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
int hyperbolic_line_type_given_point_types(
      long int pt1, long int pt2,
   int pt1_type, int pt2_type);
int hyperbolic_decide_P1(long int pt1, long int pt2);
int hyperbolic_decide_P2(long int pt1, long int pt2);
int hyperbolic_decide_P3(long int pt1, long int pt2);
int find_root_hyperbolic(
      long int rk2, int m, int verbose_level);
// m = Witt index
void find_root_hyperbolic_xyz(long int rk2, int m,
   int *x, int *y, int *z, int verbose_level);


// hyperbolic_pair_parabolic.cpp:
int parabolic_type_and_index_to_point_rk(int type,
   int index, int verbose_level);
int parabolic_even_type_and_index_to_point_rk(int type,
   int index, int verbose_level);
void parabolic_even_type1_index_to_point(int index, int *v);
void parabolic_even_type2_index_to_point(int index, int *v);
long int parabolic_odd_type_and_index_to_point_rk(long int type,
      long int index, int verbose_level);
void parabolic_odd_type1_index_to_point(long int index,
   int *v, int verbose_level);
void parabolic_odd_type2_index_to_point(long int index,
   int *v, int verbose_level);
void parabolic_point_rk_to_type_and_index(long int rk,
      long int &type, long int &index, int verbose_level);
void parabolic_even_point_rk_to_type_and_index(long int rk,
      long int &type, long int &index, int verbose_level);
void parabolic_even_point_to_type_and_index(int *v,
      long int &type, long int &index, int verbose_level);
void parabolic_odd_point_rk_to_type_and_index(long int rk,
      long int &type, long int &index, int verbose_level);
void parabolic_odd_point_to_type_and_index(int *v,
      long int &type, long int &index, int verbose_level);

void parabolic_neighbor51_odd_unrank(long int index,
```

```
   int *v, int verbose_level);
long int parabolic_neighbor51_odd_rank(int *v,
   int verbose_level);
void parabolic_neighbor52_odd_unrank(long int index,
   int *v, int verbose_level);
long int parabolic_neighbor52_odd_rank(int *v,
   int verbose_level);
void parabolic_neighbor52_even_unrank(long int index,
   int *v, int verbose_level);
long int parabolic_neighbor52_even_rank(int *v,
   int verbose_level);
void parabolic_neighbor34_unrank(long int index,
   int *v, int verbose_level);
long int parabolic_neighbor34_rank(int *v,
   int verbose_level);
void parabolic_neighbor53_unrank(long int index,
   int *v, int verbose_level);
long int parabolic_neighbor53_rank(int *v,
   int verbose_level);
void parabolic_neighbor54_unrank(long int index,
   int *v, int verbose_level);
long int parabolic_neighbor54_rank(int *v, int verbose_level);


void parabolic_unrank_line(long int &p1, long int &p2,
      long int rk, int verbose_level);
long int parabolic_rank_line(long int p1, long int p2,
      int verbose_level);
void parabolic_unrank_line_L1_even(long int &p1, long int &p2,
      long int index, int verbose_level);
long int parabolic_rank_line_L1_even(long int p1, long int p2,
   int verbose_level);
void parabolic_unrank_line_L1_odd(long int &p1, long int &p2,
      long int index, int verbose_level);
long int parabolic_rank_line_L1_odd(long int p1, long int p2,
   int verbose_level);
void parabolic_unrank_line_L2_even(long int &p1, long int &p2,
      long int index, int verbose_level);
void parabolic_unrank_line_L2_odd(long int &p1, long int &p2,
      long int index, int verbose_level);
int parabolic_rank_line_L2_even(long int p1, long int p2,
   int verbose_level);
long int parabolic_rank_line_L2_odd(long int p1, long int p2,
   int verbose_level);
void parabolic_unrank_line_L3(long int &p1, long int &p2,
      long int index, int verbose_level);
long int parabolic_rank_line_L3(long int p1, long int p2,
```

```
      int verbose_level);
void parabolic_unrank_line_L4(long int &p1, long int &p2,
      long int index, int verbose_level);
long int parabolic_rank_line_L4(long int p1, long int p2,
      int verbose_level);
void parabolic_unrank_line_L5(long int &p1, long int &p2,
      long int index, int verbose_level);
long int parabolic_rank_line_L5(long int p1, long int p2,
      int verbose_level);
void parabolic_unrank_line_L6(long int &p1, long int &p2,
      long int index, int verbose_level);
long int parabolic_rank_line_L6(long int p1, long int p2,
   int verbose_level);
void parabolic_unrank_line_L7(long int &p1, long int &p2,
      long int index, int verbose_level);
long int parabolic_rank_line_L7(long int p1, long int p2,
   int verbose_level);
void parabolic_unrank_line_L8(long int &p1, long int &p2,
      long int index, int verbose_level);
long int parabolic_rank_line_L8(long int p1, long int p2,
   int verbose_level);
long int parabolic_line_type_given_point_types(
      long int pt1, long int pt2,
      long int pt1_type, long int pt2_type, int verbose_level);
int parabolic_decide_P11_odd(long int pt1, long int pt2);
int parabolic_decide_P22_even(long int pt1, long int pt2);
int parabolic_decide_P22_odd(long int pt1, long int pt2);
int parabolic_decide_P33(long int pt1, long int pt2);
int parabolic_decide_P35(long int pt1, long int pt2);
int parabolic_decide_P45(long int pt1, long int pt2);
int parabolic_decide_P44(long int pt1, long int pt2);
void find_root_parabolic_xyz(long int rk2,
   int *x, int *y, int *z, int verbose_level);
long int find_root_parabolic(long int rk2, int verbose_level);
void parabolic_canonical_points_of_line(
   int line_type, long int pt1, long int pt2,
   long int &cpt1, long int &cpt2, int verbose_level);
void parabolic_canonical_points_L1_even(
      long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
void parabolic_canonical_points_separate_P5(
      long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
void parabolic_canonical_points_L3(
      long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
void parabolic_canonical_points_L7(
```

```
      long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
void parabolic_canonical_points_L8(
      long int pt1, long int pt2,
      long int &cpt1, long int &cpt2);
void parabolic_point_normalize(int *v, int stride, int n);
void parabolic_normalize_point_wrt_subspace(
      int *v, int stride);
void parabolic_point_properties(
      int *v, int stride, int n,
   int &f_start_with_one, int &value_middle, int &value_end,
   int verbose_level);
int parabolic_is_middle_dependent(int *vec1, int *vec2);


// hyperbolic_pair_rank_unrank.cpp
void unrank_point(int *v,
   int stride, long int rk, int verbose_level);
long int rank_point(int *v, int stride, int verbose_level);
void unrank_line(long int &p1, long int &p2,
   long int index, int verbose_level);
long int rank_line(long int p1, long int p2, int verbose_level);
int line_type_given_point_types(
      long int pt1, long int pt2,
      long int pt1_type, long int pt2_type);
long int type_and_index_to_point_rk(
      long int type,
      long int index, int verbose_level);
void point_rk_to_type_and_index(
      long int rk,
      long int &type, long int &index, int verbose_level);
void canonical_points_of_line(
      int line_type, long int pt1, long int pt2,
      long int &cpt1, long int &cpt2, int verbose_level);
void unrank_S(int *v, int stride, int m, int rk);
long int rank_S(int *v, int stride, int m);
void unrank_N(int *v, int stride, int m, long int rk);
long int rank_N(int *v, int stride, int m);
void unrank_N1(int *v, int stride, int m, long int rk);
long int rank_N1(int *v, int stride, int m);
void unrank_Sbar(int *v, int stride, int m, long int rk);
long int rank_Sbar(int *v, int stride, int m);
void unrank_Nbar(int *v, int stride, int m, long int rk);
long int rank_Nbar(int *v, int stride, int m);


};
```

```
// ##############################################################################
// orthogonal_indexing.cpp
// ##############################################################################

//! indexing of points in an orthogonal geometry O^epsilon(n,q)



class orthogonal_indexing {

public:

   quadratic_form *Quadratic_form;

   field_theory::finite_field *F;

   orthogonal_indexing();
   ~orthogonal_indexing();
   void init(quadratic_form *Quadratic_form,
         int verbose_level);
   void Q_epsilon_unrank_private(
      int *v, int stride, int epsilon, int k,
      int c1, int c2, int c3, long int a,
      int verbose_level);
   long int Q_epsilon_rank_private(
      int *v, int stride, int epsilon, int k,
      int c1, int c2, int c3,
      int verbose_level);
   //void init_hash_table_parabolic(int k, int verbose_level);
   void Q_unrank(int *v,
         int stride, int k, long int a,
         int verbose_level);
   long int Q_rank(int *v,
         int stride, int k, int verbose_level);
   void Q_unrank_directly(int *v,
         int stride, int k, long int a,
         int verbose_level);
      // parabolic quadric
      // k = projective dimension, must be even
   long int Q_rank_directly(int *v,
         int stride, int k,
         int verbose_level);
   void Qplus_unrank(int *v,
         int stride, int k, long int a,
```

```
         int verbose_level);
      // hyperbolic quadric
      // k = projective dimension, must be odd
   long int Qplus_rank(int *v,
         int stride, int k,
         int verbose_level);
   void Qminus_unrank(int *v,
         int stride, int k, long int a,
         int c1, int c2, int c3,
         int verbose_level);
      // elliptic quadric
      // k = projective dimension, must be odd
      // the form is
      // \sum_{i=0}^n x_{2i}x_{2i+1} + c1 x_{2n}^2 +
      // c2 x_{2n} x_{2n+1} + c3 x_{2n+1}^2
   long int Qminus_rank(int *v, int stride,
         int k, int c1, int c2, int c3, int verbose_level);
   void S_unrank(int *v, int stride, int n, long int a);
   void S_rank(int *v, int stride, int n, long int &a);
   void N_unrank(int *v, int stride, int n, long int a);
   void N_rank(int *v, int stride, int n, long int &a);
   void N1_unrank(int *v, int stride, int n, long int a);
   void N1_rank(int *v, int stride, int n, long int &a);
   void Sbar_unrank(int *v,
         int stride, int n, long int a, int verbose_level);
   void Sbar_rank(int *v,
         int stride, int n, long int &a, int verbose_level);
   void Nbar_unrank(int *v, int stride, int n, long int a);
   void Nbar_rank(int *v, int stride, int n, long int &a);

};


// ##############################################################################
// linear_complex.cpp
// ##############################################################################

//! a linear complex of lines in PG(3,q) under the Klein correspondence


class linear_complex {

public:

   algebraic_geometry::surface_domain *Surf;

   long int pt0_wedge;
```

```
      // in wedge coordinates 100000
   long int pt0_line;
      // pt0 = the line spanned by 1000, 0100
      // (we call it point because it is a point on the Klein quadric)
   long int pt0_klein;
      // in klein coordinates 100000

   int nb_neighbors;
      // = (q + 1) * q * (q + 1)

   long int *Neighbors; // [nb_neighbors]
      // The lines which intersect the special line.
      // In wedge ranks.
      // The array Neighbors is sorted.

   long int *Neighbor_to_line; // [nb_neighbors]
      // The lines which intersect the special line.
      // In grassmann (i.e., line) ranks.
   long int *Neighbor_to_klein; // [nb_neighbors]
      // In orthogonal ranks (i.e., points on the Klein quadric).

   linear_complex();
   ~linear_complex();
   void init(
         algebraic_geometry::surface_domain *Surf,
         int verbose_level);
   void compute_neighbors(int verbose_level);
   void make_spreadsheet_of_neighbors(
         data_structures::spreadsheet *&Sp,
      int verbose_level);

};

// ##############################################################################
// orthogonal_global.cpp
// ##############################################################################

//! global functions for orthogonal geometries


class orthogonal_global {

public:
   orthogonal_global();
   ~orthogonal_global();

   void create_FTWKB_BLT_set(orthogonal *O,
```

```
        long int *set, int *ABC, int verbose_level);
   void create_K1_BLT_set(orthogonal *O,
        long int *set, int *ABC, int verbose_level);
   void create_K2_BLT_set(orthogonal *O,
        long int *set, int *ABC, int verbose_level);
   void create_LP_37_72_BLT_set(orthogonal *O,
        long int *set, int verbose_level);
   void create_LP_37_4a_BLT_set(orthogonal *O,
        long int *set, int verbose_level);
   void create_LP_37_4b_BLT_set(orthogonal *O,
        long int *set, int verbose_level);
   void create_Law_71_BLT_set(orthogonal *O,
        long int *set, int verbose_level);
   int BLT_test_full(orthogonal *O, int size,
        long int *set, int verbose_level);
   int BLT_test(orthogonal *O, int size,
        long int *set, int verbose_level);
   int collinearity_test(orthogonal *O,
        int size, long int *set, int verbose_level);
   void plane_invariant(orthogonal *O,
      int size, int *set,
      int &nb_planes, int *&intersection_matrix,
      int &Block_size, int *&Blocks,
      int verbose_level);
   void create_Fisher_BLT_set(
        long int *Fisher_BLT, int *ABC,
        field_theory::finite_field *FQ,
        field_theory::finite_field *Fq,
        int verbose_level);
   void create_Linear_BLT_set(
        long int *BLT, int *ABC,
        field_theory::finite_field *FQ,
        field_theory::finite_field *Fq,
        int verbose_level);
   void create_Mondello_BLT_set(
        long int *BLT, int *ABC,
        field_theory::finite_field *FQ,
        field_theory::finite_field *Fq,
        int verbose_level);


};


// ##############################################################################
// orthogonal_group.cpp
// ##############################################################################
```

```
//! the group associated with O^epsilon(n,q)


class orthogonal_group {

public:

   orthogonal *O;

   quadratic_form *Quadratic_form_stack;
      // a stack of hyperbolic quadratic forms for i=1,...,m

   int *find_root_x, *find_root_y, *find_root_z;

   // stuff for Siegel_transformation
   int *Sv1, *Sv2, *Sv3, *Sv4;
   int *Gram2;
   int *ST_N1, *ST_N2, *ST_w;
   int *STr_B, *STr_Bv, *STr_w, *STr_z, *STr_x;

   orthogonal_group();
   ~orthogonal_group();
   void init(orthogonal *O, int verbose_level);

   long int find_root(
         long int rk2, int verbose_level);
   void Siegel_map_between_singular_points(
         int *T,
         long int rk_from, long int rk_to,
         long int root, int verbose_level);
   void Siegel_map_between_singular_points_hyperbolic(
         int *T,
      long int rk_from, long int rk_to,
      long int root, int m,
      int verbose_level);
   void Siegel_Transformation(int *T,
      long int rk_from, long int rk_to, long int root,
      int verbose_level);
      // root is not perp to from and to.
   void Siegel_Transformation2(int *T,
      long int rk_from, long int rk_to, long int root,
      int *B, int *Bv, int *w, int *z, int *x,
      int verbose_level);
   void Siegel_Transformation3(int *T,
      int *from, int *to, int *root,
      int *B, int *Bv, int *w, int *z, int *x,
```

```
   int verbose_level);
void random_generator_for_orthogonal_group(
   int f_action_is_semilinear,
   int f_siegel,
   int f_reflection,
   int f_similarity,
   int f_semisimilarity,
   int *Mtx, int verbose_level);
void create_random_Siegel_transformation(int *Mtx,
   int verbose_level);
   // Makes an n x n matrix only.
   // Does not put a semilinear component.
void create_random_semisimilarity(
      int *Mtx, int verbose_level);
void create_random_similarity(
      int *Mtx, int verbose_level);
   // Makes an n x n matrix only.
   // Does not put a semilinear component.
void create_random_orthogonal_reflection(int *Mtx,
   int verbose_level);
   // Makes an n x n matrix only.
   // Does not put a semilinear component.
void make_orthogonal_reflection(int *M, int *z,
   int verbose_level);
void make_Siegel_Transformation(int *M, int *v, int *u,
   int n, int *Gram, int verbose_level);
   // if u is singular and v \in \la u \ra^\perp, then
   // \pho_{u,v}(x) :=
   // x + \beta(x,v) u - \beta(x,u) v - Q(v) \beta(x,u) u
   // is called Siegel transform (see Taylor p. 148)
   // Here Q is the quadratic form and
   // \beta is the corresponding bilinear form
void Siegel_move_forward_by_index(
      long int rk1, long int rk2,
   int *v, int *w, int verbose_level);
void Siegel_move_backward_by_index(
      long int rk1, long int rk2,
   int *w, int *v, int verbose_level);
void Siegel_move_forward(
      int *v1, int *v2, int *v3, int *v4,
   int verbose_level);
void Siegel_move_backward(
      int *v1, int *v2, int *v3, int *v4,
   int verbose_level);
void move_points_by_ranks_in_place(
   long int pt_from, long int pt_to,
   int nb, long int *ranks,
```

```
      int verbose_level);
   void move_points_by_ranks(
         long int pt_from, long int pt_to,
      int nb, long int *input_ranks, long int *output_ranks,
      int verbose_level);
   void move_points(
         long int pt_from, long int pt_to,
      int nb, int *input_coords, int *output_coords,
      int verbose_level);
   void test_Siegel(int index, int verbose_level);

};


// ##############################################################################
// orthogonal.cpp
// ##############################################################################

//! an orthogonal geometry O^epsilon(n,q)


class orthogonal {

public:

   quadratic_form *Quadratic_form;

   orthogonal_indexing *Orthogonal_indexing;


   hyperbolic_pair *Hyperbolic_pair;

   field_theory::square_nonsquare *SN;

   field_theory::finite_field *F;


   int *T1, *T2, *T3; // [n * n]

   // for determine_line
   int *determine_line_v1, *determine_line_v2, *determine_line_v3;

   // for lines_on_point
   int *lines_on_point_coords1; // [alpha * n]
   int *lines_on_point_coords2; // [alpha * n]

   orthogonal *subspace;
```

```
// for perp:
long int *line_pencil; // [nb_lines]
long int *Perp1; // [alpha * (q + 1)]


orthogonal_group *Orthogonal_group;

orthogonal();
~orthogonal();
void init(int epsilon, int n,
      field_theory::finite_field *F,
   int verbose_level);
void allocate();
int evaluate_bilinear_form_by_rank(int i, int j);
void points_on_line_by_line_rank(
      long int line_rk,
   long int *line,
   int verbose_level);
void points_on_line(long int pi, long int pj,
      long int *line, int verbose_level);
void points_on_line_by_coordinates(
      long int pi, long int pj,
   int *pt_coords, int verbose_level);
void lines_on_point(long int pt,
   long int *line_pencil_point_ranks,
   int verbose_level);
void lines_on_point_by_line_rank_must_fit_into_int(
      long int pt,
      int *line_pencil_line_ranks,
      int verbose_level);
void lines_on_point_by_line_rank(
      long int pt,
   long int *line_pencil_line_ranks,
   int verbose_level);
void make_initial_partition(
      data_structures::partitionstack &S,
   int verbose_level);
void point_to_line_map(int size,
   long int *point_ranks, int *&line_vector,
   int verbose_level);
int test_if_minimal_on_line(
      int *v1, int *v2, int *v3);
void find_minimal_point_on_line(
      int *v1, int *v2, int *v3);
void zero_vector(int *u, int stride, int len);
int is_zero_vector(int *u, int stride, int len);
```

```
void change_form_value(int *u,
      int stride, int m, int multiplier);
void scalar_multiply_vector(int *u,
      int stride, int len, int multiplier);
int last_non_zero_entry(int *u, int stride, int len);
void normalize_point(int *v, int stride);
int is_ending_dependent(int *vec1, int *vec2);
void Gauss_step(int *v1, int *v2, int len, int idx);
   // afterwards: v2[idx] = 0 and v2,v1
   // span the same space as before
void perp(long int pt,
      long int *Perp_without_pt, int &sz,
   int verbose_level);
void perp_of_two_points(long int pt1,
      long int pt2, long int *Perp,
   int &sz, int verbose_level);
void perp_of_k_points(long int *pts,
      int nb_pts, long int *&Perp,
   int &sz, int verbose_level);
int triple_is_collinear(
      long int pt1, long int pt2, long int pt3);
void intersection_with_subspace(
      int *Basis, int k,
      long int *&the_points, int &nb_points,
      int verbose_level);




// orthogonal_io.cpp:
void list_points_by_type(int verbose_level);
void report_points_by_type(std::ostream &ost, int verbose_level);
void list_points_of_given_type(int t,
   int verbose_level);
void report_points_of_given_type(std::ostream &ost,
      int t, int verbose_level);
void report_points(std::ostream &ost, int verbose_level);
void report_given_point_set(std::ostream &ost,
      long int *Pts, int nb_pts, int verbose_level);
void report_lines(std::ostream &ost, int verbose_level);
void report_given_line_set(std::ostream &ost,
      long int *Lines, int nb_lines, int verbose_level);
void list_all_points_vs_points(int verbose_level);
void list_points_vs_points(int t1, int t2,
   int verbose_level);
void report(std::ostream &ost, int verbose_level);
void report_schemes(std::ostream &ost, int verbose_level);
```

```
   void report_schemes_easy(std::ostream &ost);
   void create_latex_report(int verbose_level);
   void export_incidence_matrix_to_csv(int verbose_level);
   void make_fname_incidence_matrix_csv(std::string &fname);





};


// ##############################################################################
// quadratic_form_list_coding.cpp
// ##############################################################################

//! a nondegenerate quadratic form


class quadratic_form_list_coding {

public:

   field_theory::finite_field *FQ;
   field_theory::finite_field *Fq;
   field_theory::subfield_structure *SubS;

#if 0
   int *components;
   int *embedding;
   int *pair_embedding;
      // data computed by F.subfield_embedding_2dimensional
#endif

   int alpha;
   int T_alpha;
   int N_alpha;


   int nb_terms;
   int *form_i;
   int *form_j;
   int *form_coeff;
   int *Gram;

   int r_nb_terms;
```

```
   int *r_form_i;
   int *r_form_j;
   int *r_form_coeff;
   int *r_Gram;

   int rr_nb_terms;
   int *rr_form_i;
   int *rr_form_j;
   int *rr_form_coeff;
   int *rr_Gram;

   int hyperbolic_basis[4 * 4];
   int hyperbolic_basis_inverse[4 * 4];
   int basis[4 * 4];
   int basis_subspace[2 * 2];


   int *M;



   quadratic_form_list_coding();
   ~quadratic_form_list_coding();
   void init(field_theory::finite_field *Fq,
         field_theory::finite_field *FQ,
         int f_sum_of_squares, int verbose_level);
   void print_quadratic_form_list_coded(
         int form_nb_terms,
      int *form_i, int *form_j, int *form_coeff);
   void make_Gram_matrix_from_list_coded_quadratic_form(
      int n, field_theory::finite_field &F,
      int nb_terms, int *form_i, int *form_j,
      int *form_coeff, int *Gram);
   void add_term(int n,
         field_theory::finite_field &F,
      int &nb_terms, int *form_i, int *form_j, int *form_coeff,
      int *Gram,
      int i, int j, int coeff);

};



// ##############################################################################
// quadratic_form.cpp
// ##############################################################################
```

```
//! a nondegenerate quadratic form


class quadratic_form {

public:
   int epsilon;

   int n; // the algebraic dimension

   int m; // Witt index

   int q;

   int f_even; // TRUE in characteristic two

   int form_c1, form_c2, form_c3;

   // for minus type, the form is
   // \sum_{i=0}^m x_{2i}x_{2i+1}
   // + c1 x_{2m}^2 + c2 x_{2m} x_{2m+1} + c3 x_{2m+1}^2
   // where m is the Witt index.


   std::string label_txt;
   std::string label_tex;

   ring_theory::homogeneous_polynomial_domain *Poly;
   int *the_quadratic_form;
   int *the_monomial;


   int *Gram_matrix; // [n * n]

   field_theory::finite_field *F;

   orthogonal_indexing *Orthogonal_indexing;


   quadratic_form();
   ~quadratic_form();
   void init(int epsilon, int n,
         field_theory::finite_field *F,
         int verbose_level);
   void init_form_and_Gram_matrix(int verbose_level);
   void make_Gram_matrix(int verbose_level);
```

```
   int evaluate_quadratic_form(int *v, int stride);
   int evaluate_hyperbolic_quadratic_form(
         int *v, int stride);
   int evaluate_hyperbolic_quadratic_form_with_m(
         int *v, int stride, int m);
   int evaluate_parabolic_quadratic_form(
         int *v, int stride);
   int evaluate_elliptic_quadratic_form(
         int *v, int stride);

   int evaluate_bilinear_form(int *u, int *v, int stride);
   int evaluate_hyperbolic_bilinear_form(
         int *u, int *v, int stride, int m);
   int evaluate_parabolic_bilinear_form(
         int *u, int *v, int stride, int m);
   int evaluate_bilinear_form_Gram_matrix(int *u, int *v);

   void report_quadratic_form(
         std::ostream &ost, int verbose_level);
   long int find_root(int rk2, int verbose_level);
   void Siegel_Transformation(
      int *M, int *v, int *u, int verbose_level);
   // if u is singular and v \in \la u \ra^\perp, then
   // \pho_{u,v}(x) := x + \beta(x,v) u - \beta(x,u) v - Q(v) \beta(x,u) u
   // is called the Siegel transform (see Taylor p. 148)
   // Here Q is the quadratic form
   // and \beta is the corresponding bilinear form
   void Siegel_map_between_singular_points(int *T,
         long int rk_from, long int rk_to, long int root,
      int verbose_level);
   // root is not perp to from and to.
   void choose_anisotropic_form(int verbose_level);
   void unrank_point(int *v, long int a, int verbose_level);
   long int rank_point(int *v, int verbose_level);


};


// ##############################################################################
// unusual.cpp
// ##############################################################################

//! Penttila´s unusual model to create BLT-sets


class unusual_model {
```

```
public:
   field_theory::finite_field *FQ;
   field_theory::finite_field *Fq;

   quadratic_form *Quadratic_form;

   quadratic_form_list_coding *Quadratic_form_list_coding;

   int q;
   int Q;


   unusual_model();
   ~unusual_model();
   void setup(
         field_theory::finite_field *FQ,
         field_theory::finite_field *Fq,
         int verbose_level);
   void setup2(
         field_theory::finite_field *FQ,
         field_theory::finite_field *Fq,
         int f_sum_of_squares, int verbose_level);
   void convert_to_ranks(int n,
         int *unusual_coordinates,
      long int *ranks, int verbose_level);
   void convert_from_ranks(int n,
         long int *ranks,
      int *unusual_coordinates,
      int verbose_level);
   long int convert_to_rank(
         int *unusual_coordinates,
         int verbose_level);
   void convert_from_rank(long int rank,
         int *unusual_coordinates,
      int verbose_level);
   void convert_to_usual(int n,
         int *unusual_coordinates,
      int *usual_coordinates,
      int verbose_level);
   void create_Fisher_BLT_set(long int *Fisher_BLT,
         int *ABC, int verbose_level);
   void convert_from_usual(int n,
         int *usual_coordinates,
      int *unusual_coordinates,
      int verbose_level);
   void create_Linear_BLT_set(long int *BLT,
         int *ABC, int verbose_level);
```

```
void create_Mondello_BLT_set(long int *BLT,
      int *ABC, int verbose_level);
int N2(int a);
int T2(int a);
int evaluate_quadratic_form(
      int a, int b, int c,
      int verbose_level);
int bilinear_form(
      int a1, int b1, int c1,
      int a2, int b2, int c2,
   int verbose_level);
void print_coordinates_detailed_set(
      long int *set, int len);
void print_coordinates_detailed(
      long int pt, int cnt);
int build_candidate_set(orthogonal &O, int q,
   int gamma, int delta, int m, long int *Set,
   int f_second_half, int verbose_level);
int build_candidate_set_with_offset(
      orthogonal &O, int q,
   int gamma, int delta, int offset,
   int m, long int *Set,
   int f_second_half, int verbose_level);
int build_candidate_set_with_or_without_test(
      orthogonal &O, int q,
   int gamma, int delta, int offset,
   int m, long int *Set,
   int f_second_half, int f_test,
   int verbose_level);
int create_orbit_of_psi(orthogonal &O, int q,
   int gamma, int delta, int m, long int *Set,
   int f_test, int verbose_level);
void transform_matrix_unusual_to_usual(orthogonal *O,
   int *M4, int *M5, int verbose_level);
void transform_matrix_usual_to_unusual(orthogonal *O,
   int *M5, int *M4, int verbose_level);

void parse_4by4_matrix(int *M4,
   int &a, int &b, int &c, int &d,
   int &f_semi1, int &f_semi2,
   int &f_semi3, int &f_semi4);
void create_4by4_matrix(int *M4,
   int a, int b, int c, int d,
   int f_semi1, int f_semi2,
   int f_semi3, int f_semi4,
   int verbose_level);
void print_2x2(int *v, int *f_semi);
```

```
   void print_M5(orthogonal *O, int *M5);
};




}}}




#endif /* SRC_LIB_FOUNDATIONS_ORTHOGONAL_ORTHOGONA_H_ */
```

3.20 Polish

```
// globals.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005


#ifndef ORBITER_SRC_LIB_FOUNDATIONS_GLOBALS_GLOBALS_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_GLOBALS_GLOBALS_H_




namespace orbiter {
namespace layer1_foundations {
namespace polish {


// ##############################################################################
// function_command.cpp
// ##############################################################################

//! an individual command which is part of a function expressed in reverse polish
 notation

class function_command {
public:

   int type;
   // 1 = push labeled constant
   // 2 = push immediate constant
   // 3 = push variable
   // 4 = store variable
   // 5 = mult
   // 6 = add
   // 7 = cos
   // 8 = sin
   // 9 = return
   // 10 = sqrt

   int f_has_argument;
   int arg;
   double val; // for push immediate constant
```

```
   function_command();
   ~function_command();
   void init_with_argument(int type, int arg);
   void init_push_immediate_constant(double val);
   void init_simple(int type);

};

// ##############################################################################
// function_polish_description.cpp
// ##############################################################################

//! description of a function in reverse polish notation from the command line


class function_polish_description {
public:
   int nb_constants;
   std::vector<std::string> const_names;
   std::vector<std::string> const_values;
   int nb_variables;
   std::vector<std::string> variable_names;
   int code_sz;
   std::vector<std::string> code;

   function_polish_description();
   ~function_polish_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();
};


// ##############################################################################
// function_polish.cpp
// ##############################################################################

//! a set of functions in reverse polish notation

class function_polish {
public:

   function_polish_description *Descr;

   std::vector<std::string > Variables;
```

```
   std::vector<std::pair<std::string, double> > Constants;

   std::vector<int> Entry;
   std::vector<int> Len;

   std::vector<function_command> Code;


   function_polish();
   ~function_polish();
   void init(
         function_polish_description *Descr,
         int verbose_level);
   void print_code_complete(
         int verbose_level);
   void print_code(
         int i0,  int len,
         int verbose_level);
   void evaluate(
         double *variable_values,
         double *output_values,
         int verbose_level);

};







}}}


#endif /* ORBITER_SRC_LIB_FOUNDATIONS_GLOBALS_GLOBALS_H_ */
```

3.21 Ring Theory

```
/*
 * ring_theory.h
 *
 *  Created on: Nov 1, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_FOUNDATIONS_RING_THEORY_RING_THEORY_H_
#define SRC_LIB_FOUNDATIONS_RING_THEORY_RING_THEORY_H_

namespace orbiter {
namespace layer1_foundations {
namespace ring_theory {


// ##############################################################################
// finite_ring.cpp
// ##############################################################################


//! ring of integers modulo q



class finite_ring {

   int *add_table; // [q * q]
   int *mult_table; // [q * q]

   int *f_is_unit_table; // [q]
   int *negate_table; // [q]
   int *inv_table; // [q]

   // only defined if we are a chain ring:
   int p;
   int e;
   field_theory::finite_field *Fp;

public:
   int q;

   int f_chain_ring; // true if q is a prime power



   finite_ring();
```

```
   ~finite_ring();
   void init(int q, int verbose_level);
   int get_e();
   int get_p();
   field_theory::finite_field *get_Fp();
   int zero();
   int one();
   int is_zero(int i);
   int is_one(int i);
   int is_unit(int i);
   int add(int i, int j);
   int mult(int i, int j);
   int negate(int i);
   int inverse(int i);
   int Gauss_int(int *A, int f_special,
      int f_complete, int *base_cols,
      int f_P, int *P, int m, int n, int Pn,
      int verbose_level);
      // returns the rank which is the number
      // of entries in base_cols
      // A is a m x n matrix,
      // P is a m x Pn matrix (if f_P is TRUE)
   int PHG_element_normalize(int *v, int stride, int len);
   // last unit element made one
   int PHG_element_normalize_from_front(int *v,
      int stride, int len);
   // first non unit element made one
   int PHG_element_rank(int *v, int stride, int len);
   void PHG_element_unrank(int *v, int stride, int len, int rk);
   int nb_PHG_elements(int n);
};

// ##############################################################################
// homogeneous_polynomial_domain.cpp
// ##############################################################################

//! homogeneous polynomials of a given degree in a given number of variables over
 a finite field GF(q)


class homogeneous_polynomial_domain {

private:
   enum monomial_ordering_type Monomial_ordering_type;
   field_theory::finite_field *F;
   int nb_monomials;
   int *Monomials; // [nb_monomials * nb_variables]
```

```
   std::vector<std::string> symbols;
   std::vector<std::string> symbols_latex;

   std::vector<std::string> monomial_symbols;
   std::vector<std::string> monomial_symbols_latex;
   std::vector<std::string> monomial_symbols_easy;

   int *Variables; // [nb_monomials * degree]
      // Variables contains the monomials written out
      // as a sequence of length degree
      // with entries in 0,..,nb_variables-1.
      // the entries are listed in increasing order.
      // For instance, the monomial x_0^2x_1x_3
      // is recorded as 0,0,1,3
   int nb_affine; // nb_variables^degree
   int *Affine; // [nb_affine * degree]
      // the affine elements are used for foiling
      // when doing a linear substitution
   int *v; // [nb_variables]
   int *Affine_to_monomial; // [nb_affine]
      // for each vector in the affine space,
      // record the monomial associated with it.

   int *coeff2; // [nb_monomials], used in substitute_linear
   int *coeff3; // [nb_monomials], used in substitute_linear
   int *coeff4; // [nb_monomials], used in substitute_linear
   int *factors; // [degree]
   int *my_affine; // [degree], used in substitute_linear
   int *base_cols; // [nb_monomials]
   int *type1; // [degree + 1]
   int *type2; // [degree + 1]

public:
   int q;
   int nb_variables; // number of variables
   int degree;

   homogeneous_polynomial_domain();
   ~homogeneous_polynomial_domain();
   void init(polynomial_ring_description *Descr,
         int verbose_level);
   void init(field_theory::finite_field *F,
         int nb_vars, int degree,
         monomial_ordering_type Monomial_ordering_type,
         int verbose_level);
```

```
void init_with_or_without_variables(
      field_theory::finite_field *F,
      int nb_vars, int degree,
      monomial_ordering_type Monomial_ordering_type,
      int f_has_variables,
      std::vector<std::string> *variables_txt,
      std::vector<std::string> *variables_tex,
      int verbose_level);
void print();
void print_latex(std::ostream &ost);
int get_nb_monomials();
int get_nb_variables();
field_theory::finite_field *get_F();
int get_monomial(int i, int j);
std::string &get_monomial_symbol_easy(int i);
int *get_monomial_pointer(int i);
int evaluate_monomial(int idx_of_monomial, int *coords);
void remake_symbols(int symbol_offset,
      std::string &symbol_mask, std::string &symbol_mask_latex,
      int verbose_level);
void remake_symbols_interval(int symbol_offset,
      int from, int len,
      std::string &symbol_mask, std::string &symbol_mask_latex,
      int verbose_level);
void make_monomials(
      monomial_ordering_type Monomial_ordering_type,
      int f_has_variables,
      std::vector<std::string> *variables_txt,
      std::vector<std::string> *variables_tex,
      int verbose_level);
void rearrange_monomials_by_partition_type(int verbose_level);
int index_of_monomial(int *v);
void affine_evaluation_kernel(
      int *&Kernel, int &dim_kernel, int verbose_level);
void get_quadratic_form_matrix(int *eqn, int *M);
void print_monomial(std::ostream &ost, int i);
void print_monomial(std::ostream &ost, int *mon);
void print_monomial_latex(std::ostream &ost, int *mon);
void print_monomial_latex(std::ostream &ost, int i);
void print_monomial_relaxed(std::ostream &ost, int i);
void print_monomial_latex(std::string &s, int *mon);
void print_monomial_relaxed(std::string &s, int *mon);
void print_monomial_latex(std::string &s, int i);
void print_monomial_str(std::stringstream &ost, int i);
void print_monomial_for_gap_str(std::stringstream &ost, int i);
void print_monomial_latex_str(std::stringstream &ost, int i);
void print_equation(std::ostream &ost, int *coeffs);
```

```
void print_equation_simple(std::ostream &ost, int *coeffs);
void print_equation_tex(std::ostream &ost, int *coeffs);
void print_equation_relaxed(std::ostream &ost, int *coeffs);
void print_equation_numerical(std::ostream &ost, int *coeffs);
void print_equation_lint(std::ostream &ost, long int *coeffs);
void print_equation_lint_tex(std::ostream &ost, long int *coeffs);
void print_equation_str(std::stringstream &ost, int *coeffs);
void print_equation_for_gap_str(std::stringstream &ost, int *coeffs);
void print_equation_with_line_breaks_tex(std::ostream &ost,
   int *coeffs, int nb_terms_per_line,
   const char *new_line_text);
void print_equation_with_line_breaks_tex_lint(
   std::ostream &ost, long int *coeffs, int nb_terms_per_line,
   const char *new_line_text);
void algebraic_set(int *Eqns, int nb_eqns,
      long int *Pts, int &nb_pts, int verbose_level);
void polynomial_function(int *coeff, int *f, int verbose_level);
void polynomial_function_affine(
      int *coeff, int *f, int verbose_level);
void enumerate_points(int *coeff,
      std::vector<long int> &Pts,
      int verbose_level);
void enumerate_points_lint(int *coeff,
      long int *&Pts, int &nb_pts, int verbose_level);
void enumerate_points_zariski_open_set(int *coeff,
      std::vector<long int> &Pts,
      int verbose_level);
int evaluate_at_a_point_by_rank(int *coeff, int pt);
int evaluate_at_a_point(int *coeff, int *pt_vec);
void substitute_linear(int *coeff_in, int *coeff_out,
   int *Mtx_inv, int verbose_level);
void substitute_semilinear(int *coeff_in, int *coeff_out,
   int f_semilinear, int frob_power, int *Mtx_inv,
   int verbose_level);
void substitute_line(
   int *coeff_in, int *coeff_out,
   int *Pt1_coeff, int *Pt2_coeff,
   int verbose_level);
void multiply_by_scalar(
   int *coeff_in, int scalar, int *coeff_out,
   int verbose_level);
void multiply_mod(
   int *coeff1, int *coeff2, int *coeff3,
   int verbose_level);
void multiply_mod_negatively_wrapped(
   int *coeff1, int *coeff2, int *coeff3,
   int verbose_level);
```

```
int is_zero(int *coeff);
void unrank_point(int *v, long int rk);
long int rank_point(int *v);
void unrank_coeff_vector(int *v, long int rk);
long int rank_coeff_vector(int *v);
int test_weierstrass_form(int rk,
   int &a1, int &a2, int &a3, int &a4, int &a6,
   int verbose_level);
void explore_vanishing_ideal(long int *Pts,
      int nb_pts, int verbose_level);
void vanishing_ideal(
      long int *Pts, int nb_pts, int &r, int *Kernel,
   int verbose_level);
int compare_monomials(int *M1, int *M2);
int compare_monomials_PART(int *M1, int *M2);
void print_monomial_ordering(std::ostream &ost);
int *read_from_string_coefficient_pairs(
      std::string &str, int verbose_level);
int *read_from_string_coefficient_vector(
      std::string &str, int verbose_level);
void number_of_conditions_satisfied(
      std::string &variety_label_txt,
      std::string &variety_label_tex,
      std::vector<std::string> &Variety_coeffs,
      std::string &number_of_conditions_satisfied_fname,
      std::string &label_txt,
      std::string &label_tex,
      int &nb_pts, long int *&Pts,
      int verbose_level);
void create_intersection_of_zariski_open_sets(
      std::string &variety_label_txt,
      std::string &variety_label_tex,
      std::vector<std::string> &Variety_coeffs,
      std::string &label_txt,
      std::string &label_tex,
      int &nb_pts, long int *&Pts,
      int verbose_level);
void create_projective_variety(
      std::string &variety_label,
      std::string &variety_label_tex,
      std::string &variety_coeffs,
      std::string &label_txt,
      std::string &label_tex,
      int &nb_pts, long int *&Pts,
      int verbose_level);
void create_ideal(
      std::string &ideal_label,
```

```
        std::string &ideal_label_tex,
        std::string &ideal_point_set_label,
        int &dim_kernel, int &nb_monomials, int *&Kernel,
        int verbose_level);
   void create_projective_curve(
        std::string &variety_label_txt,
        std::string &variety_label_tex,
        std::string &curve_coeffs,
        std::string &label_txt,
        std::string &label_tex,
        int &nb_pts, long int *&Pts,
        int verbose_level);
   void get_coefficient_vector(
        expression_parser::formula *Formula,
        std::string &evaluate_text,
        int *Coefficient_vector,
        int verbose_level);
   void evaluate_regular_map(
        int *Coefficient_vector,
        int nb_eqns,
        geometry::projective_space *P,
        long int *&Image_pts, int &N_points,
        int verbose_level);


};


// ##############################################################################
// longinteger_domain.cpp:
// ##############################################################################

//! domain to compute with integers of arbitrary size, using class longinteger_ob
ject

class longinteger_domain {

public:
   int compare(longinteger_object &a, longinteger_object &b);
   int compare_unsigned(longinteger_object &a, longinteger_object &b);
      // returns -1 if a < b, 0 if a = b,
      // and 1 if a > b, treating a and b as unsigned.
   int is_less_than(longinteger_object &a, longinteger_object &b);
   void subtract_signless(longinteger_object &a,
      longinteger_object &b, longinteger_object &c);
      // c = a - b, assuming a > b
   void subtract_signless_in_place(longinteger_object &a,
```

```
   longinteger_object &b);
   // a := a - b, assuming a > b
void add(longinteger_object &a,
   longinteger_object &b, longinteger_object &c);
void add_mod(longinteger_object &a,
   longinteger_object &b, longinteger_object &c,
   longinteger_object &m, int verbose_level);
void add_in_place(longinteger_object &a,
      longinteger_object &b);
   // a := a + b
void subtract_in_place(
      longinteger_object &a, longinteger_object &b);
// a := a - b
void add_int_in_place(
      longinteger_object &a, long int b);
void mult(longinteger_object &a,
   longinteger_object &b, longinteger_object &c);
void mult_in_place(
      longinteger_object &a, longinteger_object &b);
void mult_integer_in_place(longinteger_object &a, int b);
void mult_mod(longinteger_object &a,
   longinteger_object &b, longinteger_object &c,
   longinteger_object &m, int verbose_level);
void multiply_up(longinteger_object &a, int *x, int len,
      int verbose_level);
void multiply_up_lint(
      longinteger_object &a, long int *x, int len,
      int verbose_level);
int quotient_as_int(
      longinteger_object &a, longinteger_object &b);
long int quotient_as_lint(
      longinteger_object &a, longinteger_object &b);
void integral_division_exact(longinteger_object &a,
   longinteger_object &b, longinteger_object &a_over_b);
void integral_division(
   longinteger_object &a, longinteger_object &b,
   longinteger_object &q, longinteger_object &r,
   int verbose_level);
void integral_division_by_int(longinteger_object &a,
   int b, longinteger_object &q, int &r);
void integral_division_by_lint(
   longinteger_object &a,
   long int b, longinteger_object &q, long int &r);
void inverse_mod(
   longinteger_object &a,
   longinteger_object &m, longinteger_object &av,
   int verbose_level);
```

```
void extended_gcd(longinteger_object &a,
      longinteger_object &b,
   longinteger_object &g, longinteger_object &u,
   longinteger_object &v, int verbose_level);
int logarithm_base_b(longinteger_object &a, int b);
void base_b_representation(longinteger_object &a,
   int b, int *&rep, int &len);
void power_int(longinteger_object &a, int n);
void power_int_mod(longinteger_object &a, int n,
   longinteger_object &m);
void power_longint_mod(longinteger_object &a,
   longinteger_object &n, longinteger_object &m,
   int verbose_level);
void square_root(
      longinteger_object &a,
      longinteger_object &sqrt_a,
      int verbose_level);
int square_root_mod(int a, int p, int verbose_level);
   // solves x^2 = a mod p. Returns x


void create_q_to_the_n(
      longinteger_object &a, int q, int n);
void create_qnm1(longinteger_object &a, int q, int n);
void create_Mersenne(longinteger_object &M, int n);
// $M_n = 2^n - 1$
void create_Fermat(longinteger_object &F, int n);
// $F_n = 2^{2^n} + 1$
void Dedekind_number(longinteger_object &Dnq,
      int n, int q, int verbose_level);

int is_even(longinteger_object &a);
int is_odd(longinteger_object &a);
int remainder_mod_int(longinteger_object &a, int p);
int multiplicity_of_p(longinteger_object &a,
   longinteger_object &residue, int p);
long int smallest_primedivisor(
      longinteger_object &a, int p_min,
   int verbose_level);
void factor_into_longintegers(longinteger_object &a,
   int &nb_primes, longinteger_object *&primes,
   int *&exponents, int verbose_level);
void factor(longinteger_object &a, int &nb_primes,
   int *&primes, int *&exponents,
   int verbose_level);
int jacobi(longinteger_object &a, longinteger_object &m,
   int verbose_level);
```

```
   void random_number_less_than_n(longinteger_object &n,
      longinteger_object &r);
   void random_number_with_n_decimals(
      longinteger_object &R, int n, int verbose_level);
   void matrix_product(longinteger_object *A,
         longinteger_object *B,
      longinteger_object *&C, int Am, int An, int Bn);
   void matrix_entries_integral_division_exact(
         longinteger_object *A,
      longinteger_object &b, int Am, int An);
   void matrix_print_GAP(std::ostream &ost,
         longinteger_object *A,
      int Am, int An);
   void matrix_print_tex(std::ostream &ost,
         longinteger_object *A,
      int Am, int An);
   void power_mod(char *aa, char *bb, char *nn,
      longinteger_object &result, int verbose_level);
   void factorial(longinteger_object &result, int n);
   void group_order_PGL(longinteger_object &result,
      int n, int q, int f_semilinear);
   void square_root_floor(longinteger_object &a,
         longinteger_object &x, int verbose_level);
   void print_digits(char *rep, int len);
   void Chinese_Remainders(
         std::vector<long int> &Remainders,
         std::vector<long int> &Moduli,
         longinteger_object &x, longinteger_object &M,
         int verbose_level);

};



// ##############################################################################
// longinteger_object.cpp:
// ##############################################################################


//! a class to represent integers of arbitrary size


class longinteger_object {

private:
```

```
   char sgn; // TRUE if negative
   int l;
   char *r;

public:
   longinteger_object();
   ~longinteger_object();
   void freeself();

   char &ith(int i) { return r[i]; };
   char &sign() { return sgn; };
   int &len() { return l; };
   char *&rep() { return r; };
   void create(long int i, const char *file, int line);
   void create_product(int nb_factors, int *factors);
   void create_power(int a, int e);
      // creates a^e
   void create_power_minus_one(int a, int e);
      // creates a^e  - 1
   void create_from_base_b_representation(
         int b, int *rep, int len);
   void create_from_base_10_string(
         const char *str, int verbose_level);
   void create_from_base_10_string(const char *str);
   void create_from_base_10_string(std::string &str);
   int as_int();
   long int as_lint();
   void as_longinteger(longinteger_object &a);
   void assign_to(longinteger_object &b);
   void swap_with(longinteger_object &b);
   std::ostream& print(std::ostream& ost);
   std::ostream& print_not_scientific(std::ostream& ost);
   int log10();
   int output_width();
   void print_width(std::ostream& ost, int width);
   void print_to_string(std::string &s);
   void normalize();
   void negate();
   int is_zero();
   void zero();
   int is_one();
   int is_mone();
   int is_one_or_minus_one();
   void one();
   void increment();
   void decrement();
   void add_int(int a);
```

```
   void create_i_power_j(int i, int j);
   int compare_with_int(int a);
};

std::ostream& operator<<(std::ostream& ost, longinteger_object& p);





// ##############################################################################
// partial_derivative.cpp
// ##############################################################################

//! partial derivative connects two homogeneous polynomial domains


class partial_derivative {

public:
   homogeneous_polynomial_domain *H;
   homogeneous_polynomial_domain *Hd;
   int *v; // [H->get_nb_monomials()]
   int variable_idx;
   int *mapping; // [H->get_nb_monomials() * H->get_nb_monomials()]


   partial_derivative();
   ~partial_derivative();
   void init(homogeneous_polynomial_domain *H,
         homogeneous_polynomial_domain *Hd,
         int variable_idx,
         int verbose_level);
   void apply(int *eqn_in,
         int *eqn_out,
         int verbose_level);
};


// ##############################################################################
// polynomial_double_domain.cpp:
// ##############################################################################


//! domain for polynomials with coefficients of type double
```

```
class polynomial_double_domain {
public:
   int alloc_length;
   polynomial_double_domain();
   ~polynomial_double_domain();
   void init(int alloc_length);
   ring_theory::polynomial_double *create_object();
   void mult(polynomial_double *A,
         polynomial_double *B, polynomial_double *C);
   void add(polynomial_double *A,
         polynomial_double *B, polynomial_double *C);
   void mult_by_scalar_in_place(
         polynomial_double *A,
         double lambda);
   void copy(polynomial_double *A,
         polynomial_double *B);
   void determinant_over_polynomial_ring(
         polynomial_double *P,
         polynomial_double *det, int n, int verbose_level);
   void find_all_roots(polynomial_double *p,
         double *lambda, int verbose_level);
   double divide_linear_factor(polynomial_double *p,
         polynomial_double *q,
         double lambda, int verbose_level);
};


// ##############################################################################
// polynomial_double.cpp:
// ##############################################################################


//! polynomials with double coefficients, related to class polynomial_double_doma
in


class polynomial_double {
public:
   int alloc_length;
   int degree;
   double *coeff; // [alloc_length]
   polynomial_double();
   ~polynomial_double();
   void init(int alloc_length);
   void print(std::ostream &ost);
```

```
   double root_finder(int verbose_level);
   double evaluate_at(double t);
};




// ##############################################################################
// polynomial_ring_activity_description.cpp
// ##############################################################################


//! description of a polynomial ring activity

class polynomial_ring_activity_description {
public:

   int f_cheat_sheet;

   int f_ideal;
   std::string ideal_label_txt;
   std::string ideal_label_tex;
   std::string ideal_point_set_label;

   int f_apply_transformation;
   std::string apply_transformation_Eqn_in_label;
   std::string apply_transformation_vector_ge_label;

   int f_set_variable_names;
   std::string set_variable_names_txt;
   std::string set_variable_names_tex;

   int f_print_equation;
   std::string print_equation_input;


   polynomial_ring_activity_description();
   ~polynomial_ring_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};

// ##############################################################################
```

```
// polynomial_ring_description.cpp
// ##############################################################################


//! description of a polynomial ring

class polynomial_ring_description {
public:

   int f_field;
   std::string finite_field_label;

   int f_homogeneous;
   int homogeneous_of_degree;

   int f_number_of_variables;
   int number_of_variables;

   monomial_ordering_type Monomial_ordering_type;

   int f_variables;
   std::string variables_txt;
   std::string variables_tex;

   polynomial_ring_description();
   ~polynomial_ring_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};




// ##############################################################################
// ring_theory_global.cpp
// ##############################################################################

//! global functions related to ring theory


class ring_theory_global {

public:
   ring_theory_global();
```

```
~ring_theory_global();
void write_code_for_division(
      field_theory::finite_field *F,
      std::string &label_code,
      std::string &A_coeffs,
      std::string &B_coeffs,
      int verbose_level);
void polynomial_division(
      field_theory::finite_field *F,
      std::string &A_coeffs,
      std::string &B_coeffs,
      int verbose_level);
void extended_gcd_for_polynomials(
      field_theory::finite_field *F,
      std::string &A_coeffs,
      std::string &B_coeffs,
      int verbose_level);
void polynomial_mult_mod(
      field_theory::finite_field *F,
      std::string &A_coeffs,
      std::string &B_coeffs,
      std::string &M_coeffs,
      int verbose_level);
void polynomial_power_mod(
      field_theory::finite_field *F,
      std::string &A_coeffs,
      std::string &power_text,
      std::string &M_coeffs,
      int verbose_level);
void polynomial_find_roots(
      field_theory::finite_field *F,
      std::string &A_coeffs,
      int verbose_level);
void sift_polynomials(
      field_theory::finite_field *F,
      long int rk0, long int rk1, int verbose_level);
void mult_polynomials(
      field_theory::finite_field *F,
      long int rk0, long int rk1, int verbose_level);
void polynomial_division_with_report(
      field_theory::finite_field *F,
      long int rk0, long int rk1, int verbose_level);
void polynomial_division_from_file_with_report(
      field_theory::finite_field *F,
      std::string &input_file, long int rk1, int verbose_level);
void polynomial_division_from_file_all_k_error_patterns_with_report(
      field_theory::finite_field *F,
```

```
      std::string &input_file, long int rk1,
      int k, int verbose_level);
void create_irreducible_polynomial(
      field_theory::finite_field *F,
      unipoly_domain *Fq,
      unipoly_object *&Beta, int n,
      long int *cyclotomic_set, int cylotomic_set_size,
      unipoly_object *&min_poly,
      int verbose_level);
void compute_nth_roots_as_polynomials(
      field_theory::finite_field *F,
      unipoly_domain *FpX,
      unipoly_domain *Fq, unipoly_object *&Beta,
      int n1, int n2, int verbose_level);
void compute_powers(
      field_theory::finite_field *F,
      unipoly_domain *Fq,
      int n, int start_idx,
      unipoly_object *&Beta, int verbose_level);
void make_all_irreducible_polynomials_of_degree_d(
      field_theory::finite_field *F,
      int d,
      std::vector<std::vector<int> > &Table,
      int verbose_level);
int count_all_irreducible_polynomials_of_degree_d(
      field_theory::finite_field *F,
      int d, int verbose_level);
void do_make_table_of_irreducible_polynomials(
      field_theory::finite_field *F,
      int deg, int verbose_level);
char *search_for_primitive_polynomial_of_given_degree(
      int p,
   int degree, int verbose_level);
void search_for_primitive_polynomials(
      int p_min, int p_max,
   int n_min, int n_max, int verbose_level);
void factor_cyclotomic(
      int n, int q, int d,
   int *coeffs, int f_poly,
   std::string &poly, int verbose_level);
void oval_polynomial(
      field_theory::finite_field *F,
   int *S, unipoly_domain &D, unipoly_object &poly,
   int verbose_level);
void print_longinteger_after_multiplying(
      std::ostream &ost,
      int *factors, int len);
```

```
};


// ##############################################################################
// table_of_irreducible_polynomials.cpp
// ##############################################################################

//! a table of all irreducible polynomials over a finite field of bounded degree

class table_of_irreducible_polynomials {
public:
   int k;
   int q;
   field_theory::finite_field *F;
   int nb_irred;
   int *Nb_irred;
   int *First_irred;
   int **Tables;
   int *Degree;

   table_of_irreducible_polynomials();
   ~table_of_irreducible_polynomials();
   void init(int k,
         field_theory::finite_field *F,
         int verbose_level);
   void print(std::ostream &ost);
   void print_polynomials(std::ostream &ost);
   int select_polynomial_first(
         int *Select, int verbose_level);
   int select_polynomial_next(
         int *Select, int verbose_level);
   int is_irreducible(unipoly_object &poly, int verbose_level);
   void factorize_polynomial(
         unipoly_object &char_poly, int *Mult,
      int verbose_level);
};


// ##############################################################################
// unipoly_domain.cpp:
// ##############################################################################

//! domain of polynomials in one variable over a finite field

class unipoly_domain {
```

```
private:
   field_theory::finite_field *F;
   int f_factorring;
   int factor_degree;
   int *factor_coeffs; // [factor_degree + 1]
   unipoly_object factor_poly;
      // the coefficients of factor_poly are negated
      // so that mult_mod is easier

   int f_print_sub;
   //int f_use_variable_name;
   std::string variable_name;

public:

   unipoly_domain();
   unipoly_domain(
         field_theory::finite_field *GFq);
   void init_basic(
         field_theory::finite_field *F, int verbose_level);
   unipoly_domain(
         field_theory::finite_field *GFq,
         unipoly_object m,
         int verbose_level);
   ~unipoly_domain();
   void init_variable_name(std::string &label);
   void init_factorring(
         field_theory::finite_field *F,
         unipoly_object m,
         int verbose_level);
   field_theory::finite_field *get_F();
   int &s_i(unipoly_object p, int i)
      { int *rep = (int *) p; return rep[i + 1]; };
   void create_object_of_degree(
         unipoly_object &p, int d);
   void create_object_of_degree_no_test(
         unipoly_object &p, int d);
   void create_object_of_degree_with_coefficients(
         unipoly_object &p,
      int d, int *coeff);
   void create_object_by_rank(
         unipoly_object &p, long int rk,
         const char *file, int line, int verbose_level);
   void create_object_from_csv_file(
      unipoly_object &p, std::string &fname,
      const char *file, int line,
      int verbose_level);
```

```
void create_object_by_rank_longinteger(
      unipoly_object &p,
   longinteger_object &rank,
   const char *file, int line,
   int verbose_level);
void create_object_by_rank_string(
   unipoly_object &p, std::string &rk, int verbose_level);
void create_Dickson_polynomial(
      unipoly_object &p, int *map);
void delete_object(unipoly_object &p);
void unrank(unipoly_object p, int rk);
void unrank_longinteger(
      unipoly_object p, longinteger_object &rank);
int rank(unipoly_object p);
void rank_longinteger(
      unipoly_object p, longinteger_object &rank);
int degree(unipoly_object p);
void print_object(unipoly_object p, std::ostream &ost);
void print_object_sstr(
      unipoly_object p, std::stringstream &ost);
void print_object_tight(
      unipoly_object p, std::ostream &ost);
void print_object_sparse(
      unipoly_object p, std::ostream &ost);
void print_object_dense(
      unipoly_object p, std::ostream &ost);
void assign(
      unipoly_object a, unipoly_object &b,
      int verbose_level);
void one(unipoly_object p);
void m_one(unipoly_object p);
void zero(unipoly_object p);
int is_one(unipoly_object p);
int is_zero(unipoly_object p);
void negate(unipoly_object a);
void make_monic(unipoly_object &a);
void add(unipoly_object a,
      unipoly_object b,
      unipoly_object &c);
void mult(unipoly_object a,
      unipoly_object b,
      unipoly_object &c,
      int verbose_level);
void mult_mod(unipoly_object a,
   unipoly_object b,
   unipoly_object &c,
   unipoly_object m,
```

```
   int verbose_level);
void mult_mod_negated(unipoly_object a,
      unipoly_object b,
      unipoly_object &c,
   int factor_polynomial_degree,
   int *factor_polynomial_coefficients_negated,
   int verbose_level);
void Frobenius_matrix_by_rows(int *&Frob,
   unipoly_object factor_polynomial,
   int verbose_level);
   // the j-th row of Frob is x^{j*q} mod m
void Frobenius_matrix(int *&Frob,
      unipoly_object factor_polynomial,
   int verbose_level);
void Berlekamp_matrix(int *&B,
      unipoly_object factor_polynomial,
   int verbose_level);
void exact_division(unipoly_object a,
   unipoly_object b, unipoly_object &q,
   int verbose_level);
void division_with_remainder(
      unipoly_object a, unipoly_object b,
   unipoly_object &q, unipoly_object &r,
   int verbose_level);
void derivative(unipoly_object a, unipoly_object &b);
int compare_euclidean(unipoly_object m, unipoly_object n);
void greatest_common_divisor(
      unipoly_object m, unipoly_object n,
   unipoly_object &g, int verbose_level);
void extended_gcd(
      unipoly_object m, unipoly_object n,
   unipoly_object &u, unipoly_object &v,
   unipoly_object &g, int verbose_level);
int is_squarefree(unipoly_object p,
      int verbose_level);
void compute_normal_basis(int d,
      int *Normal_basis,
   int *Frobenius, int verbose_level);
void order_ideal_generator(int d,
      int idx, unipoly_object &mue,
   int *A, int *Frobenius,
   int verbose_level);
   // Lueneburg~\cite{Lueneburg87a} p. 105.
   // Frobenius is a matrix of size d x d
   // A is a matrix of size (d + 1) x d
void matrix_apply(unipoly_object &p, int *Mtx, int n,
   int verbose_level);
```

```
   // The matrix is applied on the left
void substitute_matrix_in_polynomial(unipoly_object &p,
   int *Mtx_in, int *Mtx_out, int k, int verbose_level);
   // The matrix is substituted into the polynomial
int substitute_scalar_in_polynomial(unipoly_object &p,
   int scalar, int verbose_level);
   // The scalar ´scalar´ is substituted into the polynomial
void module_structure_apply(int *v, int *Mtx, int n,
   unipoly_object p, int verbose_level);
void take_away_all_factors_from_b(unipoly_object a,
   unipoly_object b, unipoly_object &a_without_b,
   int verbose_level);
   // Computes the polynomial $r$ with
   //\begin{enumerate}
   //\item
   //$r$ divides $a$
   //\item
   //$gcd(r,b) = 1$ and
   //\item
   //each irreducible polynomial dividing $a/r$ divides $b$.
   //Lueneburg~\cite{Lueneburg87a}, p. 37.
   //\end{enumerate}
int is_irreducible(unipoly_object a, int verbose_level);
void singer_candidate(unipoly_object &m,
   int p, int d, int b, int a);
int is_primitive(unipoly_object &m,
   longinteger_object &qm1,
   int nb_primes, longinteger_object *primes,
   int verbose_level);
void get_a_primitive_polynomial(unipoly_object &m,
   int f, int verbose_level);
void get_an_irreducible_polynomial(unipoly_object &m,
   int f, int verbose_level);
void power_int(unipoly_object &a,
      long int n, int verbose_level);
void power_longinteger(unipoly_object &a,
      longinteger_object &n,
      int verbose_level);
void power_mod(unipoly_object &a, unipoly_object &m,
      long int n, int verbose_level);
void power_coefficients(unipoly_object &a, int n);
void minimum_polynomial(unipoly_object &a,
   int alpha, int p, int verbose_level);
int minimum_polynomial_factorring(int alpha, int p,
   int verbose_level);
void minimum_polynomial_factorring_longinteger(
   longinteger_object &alpha,
```

```
   longinteger_object &rk_minpoly,
   int p, int verbose_level);
void print_vector_of_polynomials(
      unipoly_object *sigma, int deg);
void minimum_polynomial_extension_field(
      unipoly_object &g,
   unipoly_object m,
   unipoly_object &minpol, int d, int *Frobenius,
   int verbose_level);
   // Lueneburg~\cite{Lueneburg87a}, p. 112.
void characteristic_polynomial(int *Mtx, int k,
   unipoly_object &char_poly, int verbose_level);
void print_matrix(
      unipoly_object *M, int k);
void determinant(unipoly_object *M,
      int k, unipoly_object &p,
   int verbose_level);
void deletion_matrix(unipoly_object *M,
      int k, int delete_row,
   int delete_column, unipoly_object *&N,
   int verbose_level);
void center_lift_coordinates(unipoly_object a, int q);
void reduce_modulo_p(unipoly_object a, int p);

//unipoly_domain2.cpp:
void mult_easy(unipoly_object a,
      unipoly_object b,
      unipoly_object &c);
void print_coeffs_top_down_assuming_one_character_per_digit(
      unipoly_object a, std::ostream &ost);
void print_coeffs_top_down_assuming_one_character_per_digit_with_degree_given(
      unipoly_object a, int m, std::ostream &ost);
void mult_easy_with_report(
      long int rk_a, long int rk_b, long int &rk_c,
      std::ostream &ost, int verbose_level);
void division_with_remainder_from_file_with_report(
      std::string &input_fname, long int rk_b,
      long int &rk_q, long int &rk_r,
      std::ostream &ost, int verbose_level);
void division_with_remainder_from_file_all_k_bit_error_patterns(
      std::string &input_fname, long int rk_b, int k,
      long int *&rk_q, long int *&rk_r, int &n, int &N,
      std::ostream &ost, int verbose_level);
void division_with_remainder_numerically_with_report(
      long int rk_a, long int rk_b,
      long int &rk_q, long int &rk_r,
      std::ostream &ost, int verbose_level);
```

```
   void division_with_remainder_with_report(
         unipoly_object &a, unipoly_object &b,
         unipoly_object &q, unipoly_object &r,
         int f_report, std::ostream &ost,
         int verbose_level);


};

}}}


#endif /* SRC_LIB_FOUNDATIONS_RING_THEORY_RING_THEORY_H_ */
```

3.22 Solvers

```
// solvers.h
//
// Anton Betten
//
// moved here from galois.h: July 27, 2018
// started as orbiter:  October 23, 2002
// 2nd version started:  December 7, 2003
// galois started:  August 12, 2005


#ifndef ORBITER_SRC_LIB_FOUNDATIONS_SOLVERS_SOLVERS_H_
#define ORBITER_SRC_LIB_FOUNDATIONS_SOLVERS_SOLVERS_H_


namespace orbiter {
namespace layer1_foundations {
namespace solvers {


// ##############################################################################
// diophant_activity_description.cpp
// ##############################################################################


//! to describe an activity for a diophantine system from command line arguments

class diophant_activity_description {
public:

   int f_input_file;
   std::string input_file;
   int f_print;
   int f_solve_mckay;
   int f_solve_standard;
   int f_solve_DLX;

   int f_draw_as_bitmap;
   int box_width;
   int bit_depth; // 8 or 24
   int f_draw;
   int f_perform_column_reductions;

   int f_project_to_single_equation_and_solve;
   int eqn_idx;
   int solve_case_idx;
```

```
   int f_project_to_two_equations_and_solve;
   int eqn1_idx;
   int eqn2_idx;
   int solve_case_idx_r;
   int solve_case_idx_m;

   int f_test_single_equation;
   int max_number_of_coefficients;



   diophant_activity_description();
   ~diophant_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();



};


// ##############################################################################
// diophant_activity.cpp
// ##############################################################################


//! to perform an activity for a diophantine system using diophant_activity_descr
iption

class diophant_activity {
public:

   diophant_activity_description *Descr;

   diophant_activity();
   ~diophant_activity();
   void init_from_file(diophant_activity_description *Descr,
         int verbose_level);
   void perform_activity(
         diophant_activity_description *Descr, diophant *Dio,
         int verbose_level);


};
```

```
// ##############################################################################
// diophant_create.cpp
// ##############################################################################


//! to create a diophantine system from diophant_description

class diophant_create {
public:

   diophant_description *Descr;

   diophant *D;

   diophant_create();
   ~diophant_create();
   void init(
         diophant_description *Descr,
         int verbose_level);

};



// ##############################################################################
// diophant_description.cpp
// ##############################################################################


//! to describe a diophantine system from command line arguments

class diophant_description {
public:
   int f_q;
   int input_q;
   int f_override_polynomial;
   std::string override_polynomial;

   int f_maximal_arc;
   int maximal_arc_sz;
   int maximal_arc_d;
   std::string maximal_arc_secants_text;
   std::string external_lines_as_subset_of_secants_text;

   int f_label;
```

```
   std::string label;

   int f_coefficient_matrix;
   std::string coefficient_matrix_label;

   int f_problem_of_Steiner_type;
   int problem_of_Steiner_type_nb_t_orbits;
   std::string problem_of_Steiner_type_covering_matrix_fname;

   int f_coefficient_matrix_csv;
   std::string coefficient_matrix_csv;


   int f_RHS;
   std::string RHS_text;

   int f_RHS_csv;
   std::string RHS_csv_text;

   int f_RHS_constant;
   std::string RHS_constant_text;

   int f_x_max_global;
   int x_max_global;

   int f_x_min_global;
   int x_min_global;

   int f_x_bounds;
   std::string x_bounds_text;

   int f_x_bounds_csv;
   std::string x_bounds_csv;

   int f_has_sum;
   int has_sum;


   diophant_description();
   ~diophant_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};
```

```
// ##############################################################################
// diophant.cpp
// ##############################################################################


//! diophantine systems of equations (i.e., linear systems over the integers)

//! there are three types of conditions:
//! t_EQ: equality, the sum in row i on the left must equal RHS[i]
//! t_LE: inequality, the sum in row i on the left must
//!         be less than or equal to RHS[i]
//! t_INT: interval, the sum in row i on the left must
//!         be in the closed interval RHS_low[i] and RHS[i].
//! t_ZOR: Zero or otherwise: the sum in row i on the left must
//!         be zero or equal to RHS[i]
//! Here, the sum on the left in row i means
//! the value \sum_{j=0}^{n-1} A(i,j) * x[j].



class diophant {
public:
   std::string label;
   int m; // number of equations or inequalities
   int n; // number of variables
   int f_has_sum;
   int sum; // constraint: sum(i=0..(n-1); x[i]) = sum
   int sum1;
   //int f_x_max;
   // with constraints: x[i] <= x_max[i] for i=0..(n-1)

   int *A; // [m * n] the coefficient matrix
   int *G; // [m * n] matrix of gcd values
   int *x_max; // [n] upper bounds for x
   int *x_min; // [n] lower bounds for x
   int *x; // [n]  current value of x
   int *RHS; // [m] the right hand sides
   int *RHS_low; // [m] the minimum for the right hand side
      // this is used only in the McKay solver
   int *RHS1;
      // [m] he current values of the RHS
      // (=RHS - what is chosen on the left
   diophant_equation_type *type;
   char **eqn_label; // [m] a label for each equation / inequality

   int f_has_var_labels;
```

```
int *var_labels; // [n]


// the following vectors are used by diophant::test_solution
int *X; // [n]
int *Y; // [m]

std::deque<std::vector<int> > _results;
int _maxresults;
int _resultanz;
int _cur_result;
long int nb_steps_betten;
int f_max_time;
int f_broken_off_because_of_maxtime;
int max_time_in_sec;
int max_time_in_ticks;
int t0;


diophant();
~diophant();

void open(int m, int n);
void init_var_labels(
      long int *labels, int verbose_level);
void join_problems(
      diophant *D1, diophant *D2,
      int verbose_level);
void init_partition_problem(
   int *weights, int nb_weights, int target_value,
   int verbose_level);
void init_partition_problem_with_bounds(
   int *weights, int *bounds,
   int nb_weights, int target_value,
   int verbose_level);
void init_problem_of_Steiner_type_with_RHS(
      int nb_rows,
   int nb_cols, int *Inc, int nb_to_select,
   int *Rhs, int verbose_level);
void init_problem_of_Steiner_type(
      int nb_rows, int nb_cols,
   int *Inc, int nb_to_select, int verbose_level);
void init_RHS(
      int RHS_value, int verbose_level);
void init_clique_finding_problem(int *Adj, int nb_pts,
   int nb_to_select, int verbose_level);
void fill_coefficient_matrix_with(int a);
```

```
void set_x_min_constant(int a);
void set_x_max_constant(int a);
int &Aij(int i, int j);
int &Gij(int i, int j);
int &RHSi(int i);
int &RHS_low_i(int i);
void init_eqn_label(int i, char *label);
void print();
void print_tight();
void print_dense();
void print2(int f_with_gcd);
void print_compressed();
void print_eqn(int i, int f_with_gcd);
void print_eqn_compressed(int i);
void print_eqn_dense(int i);
void print_x_long();
void print_x(int header);
int RHS_ge_zero();
int solve_first(int verbose_level);
int solve_next();
int solve_first_mckay(
      int f_once, int verbose_level);
void write_solutions(std::string &fname, int verbose_level);
void read_solutions_from_file(std::string &fname_sol,
   int verbose_level);
void get_solutions(long int *&Sol, int &nb_sol, int verbose_level);
void get_solutions_full_length(int *&Sol, int &nb_sol,
   int verbose_level);
void test_solution_full_length(int *sol, int verbose_level);
int solve_all_DLX(int verbose_level);
int solve_all_DLX_with_RHS(
      int f_write_tree, const char *fname_tree,
   int verbose_level);
int solve_all_DLX_with_RHS_and_callback(
      int f_write_tree,
   const char *fname_tree,
   void (*user_callback_solution_found)(int *sol, int len,
      int nb_sol, void *data),
   int verbose_level);
int solve_all_mckay(
      long int &nb_backtrack_nodes, int maxresults,
      int verbose_level);
int solve_once_mckay(int verbose_level);
int solve_all_betten(int verbose_level);
int solve_all_betten_with_conditions(int verbose_level,
   int f_max_sol, int max_sol,
   int f_max_time, int max_time_in_seconds);
```

```
int solve_first_betten(int verbose_level);
int solve_next_mckay(int verbose_level);
int solve_next_betten(int verbose_level);
int j_fst(int j, int verbose_level);
int j_nxt(int j, int verbose_level);
void solve_mckay(
      const char *label, int maxresults,
   long int &nb_backtrack_nodes, int &nb_sol,
   int verbose_level);
void solve_mckay_override_minrhs_in_inequalities(
      const char *label,
   int maxresults, long int &nb_backtrack_nodes,
   int minrhs, int &nb_sol, int verbose_level);
void latex_it();
void latex_it(std::ostream &ost);
void trivial_row_reductions(
      int &f_no_solution, int verbose_level);
diophant *trivial_column_reductions(int verbose_level);
int count_non_zero_coefficients_in_row(int i);
void coefficient_values_in_row(int i, int &nb_values,
   int *&values, int *&multiplicities, int verbose_level);
int maximum_number_of_non_zero_coefficients_in_row();
void get_coefficient_matrix(
      int *&M, int &nb_rows, int &nb_cols,
   int verbose_level);
void save_in_general_format(
      std::string &fname, int verbose_level);
void read_general_format(
      std::string &fname, int verbose_level);
void eliminate_zero_rows_quick(int verbose_level);
void eliminate_zero_rows(
      int *&eqn_number, int verbose_level);
int is_zero_outside(int first, int len, int i);
void project(
      diophant *D, int first, int len, int *&eqn_number,
   int &nb_eqns_replaced, int *&eqns_replaced,
   int verbose_level);
void split_by_equation(
      int eqn_idx, int f_solve_case,
      int solve_case_idx, int verbose_level);
void split_by_two_equations(
      int eqn1_idx, int eqn2_idx,
      int f_solve_case,
      int solve_case_idx_r, int solve_case_idx_m,
      int verbose_level);
void project_to_single_equation_and_solve(
      int max_number_of_coefficients,
```

```
      int verbose_level);
void project_to_single_equation(
      diophant *D, int eqn_idx,
      int verbose_level);
void project_to_two_equations(
      diophant *D, int eqn1_idx, int eqn2_idx,
      int verbose_level);
void multiply_A_x_to_RHS1();
void write_xml(
      std::ostream &ost, const char *label);
void read_xml(
      std::ifstream &f, char *label, int verbose_level);
   // label will be set to the label that is in the file
   // therefore, label must point to sufficient memory
void append_equation();
void delete_equation(int I);
void write_gurobi_binary_variables(const char *fname);
void draw_as_bitmap(std::string &fname,
      int f_box_width, int box_width,
      int bit_depth, int verbose_level);
void draw_it(
      std::string &fname_base,
      graphics::layered_graph_draw_options *Draw_options,
      int verbose_level);
void draw_partitioned(
      std::string &fname_base,
      graphics::layered_graph_draw_options *Draw_options,
   int f_solution, int *solution, int solution_sz,
   int verbose_level);
int test_solution(int *sol, int len, int verbose_level);
void get_columns(int *col, int nb_col,
      data_structures::set_of_sets *&S,
   int verbose_level);
void test_solution_file(std::string &solution_file,
   int verbose_level);
void analyze(int verbose_level);
int is_of_Steiner_type();
void make_clique_graph_adjacency_matrix(
      data_structures::bitvector *&Adj,
   int verbose_level);
void make_clique_graph(
      graph_theory::colored_graph *&CG,
      int verbose_level);
void make_clique_graph_and_save(
      std::string &clique_graph_fname,
   int verbose_level);
void test_if_the_last_solution_is_unique();
```

```
   int solve_first_mckay_once_option(
         int f_once, int verbose_level);


};


// ##############################################################################
// dlx_problem_description.cpp
// ##############################################################################


//! description of a problem instance for dancing links solver


class dlx_problem_description {
public:

   int f_label_txt;
   std::string label_txt;
   int f_label_tex;
   std::string label_tex;

   int f_data_label;
   std::string data_label;

   int f_data_matrix;
   int *data_matrix;
   int data_matrix_m;
   int data_matrix_n;

   int f_write_solutions;
   int f_write_tree;

   int f_tracking_depth;
   int tracking_depth;


   dlx_problem_description();
   ~dlx_problem_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();
```

```
};



// ##############################################################################
// dlx_solver.cpp
// ##############################################################################


//! An implementation of Donald Knuth´s dancing links algorithm to solve exact co
ver problems



class dlx_solver {
public:

   dlx_problem_description *Descr;

   int *Input_data;
   int nb_rows;
   int nb_cols;

   std::string solutions_fname;
   std::ofstream *fp_sol;

   std::string tree_fname;
   int write_tree_cnt;

   std::ofstream *fp_tree;


   int nRow;
   int nCol;

   int f_has_RHS; // [nCol]
   int *target_RHS; // [nCol]
   int *current_RHS; // [nCol]
   int *current_row; // [nCol]
   int *current_row_save; // [sum_rhs]


   // we allow three types of conditions:
   // equations t_EQ
   // inequalities t_LE
   // zero or a fixed value t_ZOR
```

```
int f_type;
diophant_equation_type *type; // [nCol]
int *changed_type_columns; // [nCol]
int *nb_changed_type_columns; // [sum_rhs]
int nb_changed_type_columns_total;

int *Result; // [nRow]
int *Nb_choices; // [nRow]
int *Cur_choice; // [nRow]
int *Cur_col; // [nRow]
int *Nb_col_nodes; // [nCol]
int nb_sol;
int nb_backtrack_nodes;
dlx_node *Matrix; // [nRow * nCol]
dlx_node *Root;

int f_has_callback_solution_found;
void (*callback_solution_found)(
      int *solution, int len, int nb_sol, void *data);
void *callback_solution_found_data;



dlx_solver();
~dlx_solver();
void init(
      dlx_problem_description *Descr,
      int verbose_level);
int dataLeft(int i);
int dataRight(int i);
int dataUp(int i);
int dataDown(int i);
void install_callback_solution_found(
   void (*callback_solution_found)(
         int *solution, int len, int nb_sol, void *data),
   void *callback_solution_found_data);
void de_install_callback_solution_found();
void Test();
void TransposeAppendAndSolve(int *Data, int nb_rows, int nb_cols,
   int verbose_level);
void TransposeAndSolveRHS(int *Data, int nb_rows, int nb_cols,
   int *RHS, int f_has_type, diophant_equation_type *type,
   int verbose_level);
void AppendRowAndSolve(int *Data, int nb_rows, int nb_cols,
   int verbose_level);
void AppendRowAndSolveRHS(int *Data, int nb_rows, int nb_cols,
   int *RHS, int f_has_type, diophant_equation_type *type,
```

```
      int verbose_level);
   void Solve(int verbose_level);
   void Solve_with_RHS(int *RHS, int f_has_type,
         diophant_equation_type *type,
      int verbose_level);
   void open_solution_file(int verbose_level);
   void close_solution_file();
   void open_tree_file(int verbose_level);
   void close_tree_file();
   void Create_RHS(int nb_cols, int *RHS, int f_has_type,
         diophant_equation_type *type, int verbose_level);
   void Delete_RHS();
   void CreateMatrix(int *Data,
         int nb_rows, int nb_cols, int verbose_level);
   void DeleteMatrix();
   dlx_node *get_column_header(int c);
   dlx_node *ChooseColumnFancy();
   dlx_node *ChooseColumn();
   dlx_node *ChooseColumnFancyRHS();
   dlx_node *ChooseColumnRHS();
   void write_tree(int k);
   void print_if_necessary(int k);
   void process_solution(int k);
   void count_nb_choices(int k, dlx_node *Column);
   int IsDone();
   int IsColumnDone(int c);
   int IsColumnNotDone(int c);
   void Search(int k);
   void SearchRHS(int k, int verbose_level);
   void Cover(dlx_node *ColNode);
   void UnCover(dlx_node *ColNode);



};

//! internal class for the dancing links exact cover algorithm

struct dlx_node {

    dlx_node * Header;

    dlx_node * Left;
    dlx_node * Right;
    dlx_node * Up;
    dlx_node * Down;
```

```
    int row; // row index
    int col; // col index
};
```

```
// ##############################################################################
// mckay.cpp
// ##############################################################################


#define MCKAY_DEBUG


//! Possolve is a solver for systems of diophantine equations due to Brendan McKa
y (ca 1997)


namespace mckay {


   #include <stdio.h>
   #include <math.h>


   //#undef MCKAY_DEBUG
   #define INTERVAL_IN_SECONDS 1

   //! a term in a diophantine system of type tMCKAY

   typedef struct {int var,coeff;} term;
   typedef std::vector<term> equation;

   //! solves diophantine systems according to McKay

   class tMCKAY {
   public:
      tMCKAY();
      void Init(diophant *lgs, const char *label,
         int aEqnAnz, int aVarAnz);
      void possolve(std::vector<int> &lo, std::vector<int> &hi,
```

```
            std::vector<equation> &eqn,
            std::vector<int> &lorhs, std::vector<int> &hirhs,
            std::vector<int> &neqn, int numeqn, int numvar,
         int verbose_level);

      long int nb_calls_to_solve;
      int first_moved;
      int second_moved;
      const char *problem_label;

   protected:
      bool subtract(int eqn1, equation &e1, int l1, int lors1,
         int hirs1, int eqn2, equation &e2, int *pl2,
         int *plors2, int *phirs2, int verbose_level);
      void pruneqn(std::vector<int> &lo, std::vector<int> &hi,
            int numvar,
            std::vector<int> &lorhs, std::vector<int> &hirhs,
            std::vector<equation> &eqn, std::vector<int> &neqn,
            int numeqn, int verbose_level);
      void varprune(std::vector<int> &lo, std::vector<int> &hi,
            std::vector<int> &lorhs, std::vector<int> &hirhs,
            std::vector<equation> &eqn, std::vector<int> &neqn,
         int numeqn, int verbose_level);
      void puteqns(std::vector<int> &lo, std::vector<int> &hi,
         int numvar,
         std::vector<int> &lorhs, std::vector<int> &hirhs,
         std::vector<equation> &eqn, std::vector<int> &neqn,
         int numeqn);
      int divideeqns(std::vector<int> &lorhs, std::vector<int> &hirhs,
            std::vector<equation> &eqn, std::vector<int> &neqn,
         int numeqn);
      int gcd(int n1,int n2);
      void solve(int level,
            std::vector<int> &alo, std::vector<int> &ahi,
            std::vector<bool> &aactive, int numvar,
            std::vector<int> &lorhs, std::vector<int> &hirhs,
            std::vector<equation> &eqn, std::vector<int> &neqn,
         int numeqn, int verbose_level);
      int restrict_variables(int level,
            std::vector<int> &lo, std::vector<int> &hi,
            std::vector<bool> &active, int numvar,
            std::vector<int> &lorhs, std::vector<int> &hirhs,
            std::vector<equation> &eqn, std::vector<int> &neqn,
         int numeqn, int &f_restriction_made,
         int verbose_level);
      void log_12l(long int current_node, int level);
```

```
      int _eqnanz;
      int _varanz;
      std::vector<bool> unitcoeffs;
      std::vector<bool> active;
      int rekurs;
      bool _break;

      diophant *D;

   #ifdef MCKAY_DEBUG
      std::vector<int> range,split,branch;
      int ticks0;
   #endif

   };
}


}}}



#endif /* ORBITER_SRC_LIB_FOUNDATIONS_SOLVERS_SOLVERS_H_ */
```

Chapter 4

Layer 2 – Legacy Code: Discreta Project

4.1 Discreta – Typed Objects

```
// discreta.h
//
// Anton Betten
//
// started:  18.12.1998
// modified: 23.03.2000
// moved from D2 to ORBI Nov 15, 2007


#ifndef ORBITER_SRC_LIB_DISCRETA_DISCRETA_H_
#define ORBITER_SRC_LIB_DISCRETA_DISCRETA_H_



#pragma once

#include <iostream>
#include <fstream>
#include <sstream>

#include <stdlib.h>
#include <string.h>

#include "layer1_foundations/foundations.h"

using namespace orbiter;
using namespace orbiter::layer1_foundations;
```

```
namespace orbiter {

//! legacy project DISCRETA provides typed objects

namespace layer2_discreta {

//! typed objects

namespace typed_objects {





#define NB_BITS_THRESHOLD_FOR_LONGINTEGER 32
#define SAVE_ASCII_USE_COMPRESS

#define FITS_INTO_ONE_BYTE(a) (((a) > -126) && ((a) < 127))




#define NOT_EXISTING_FUNCTION(s)  cout << "The function " << s << " does not exis
t in this class\n";


enum kind {
   BASE = 0,
   INTEGER = 1,
   VECTOR = 2,
   NUMBER_PARTITION = 3,
   // RATIONAL /* BRUCH */ = 4,
   PERMUTATION = 6,


   MATRIX = 11,

   LONGINTEGER = 22,

   MEMORY = 39,

   HOLLERITH = 44,

   DATABASE = 50,
   BTREE = 51,
```

```
   PERM_GROUP = 56,
   PERM_GROUP_STAB_CHAIN = 57,

   BT_KEY = 61,

   DESIGN_PARAMETER = 70,

   UNIPOLY = 79,

   DESIGN_PARAMETER_SOURCE = 83,

   BITMATRIX = 90,

};

enum domain_type {
   GFp = 1,
   GFq = 2,
   Orbiter_finite_field = 3
};

enum action_kind {
   vector_entries = 1,
   vector_positions = 2
};

enum actionkind {
   on_sets,
   on_subset_of_group_elements_by_conjugation,
   on_subset_of_group_elements_by_conjugation_with_table,
   on_group_elements_via_conjugation_using_group_table,
   on_points
};

enum numeric_mult_type {
   with_perm_group,
   with_group_table
};

enum printing_mode_enum {
   printing_mode_ascii,
   printing_mode_latex,
   printing_mode_ascii_file,
   printing_mode_gap
};
```

```
enum bt_key_kind {
   bt_key_int = 0,
   bt_key_string = 1,
   bt_key_int_vec = 2
};

enum design_parameter_rule {
   rule_complementary = 1,
   rule_reduced_t = 2,
   rule_derived = 3,
   rule_residual = 4,
   rule_alltop = 5,
   rule_supplementary_reduced_t = 6,
   rule_supplementary_derived = 7,
   rule_supplementary_residual = 8,
   rule_supplementary_alltop = 9,
   rule_trung_complementary = 10,
   rule_supplementary = 11,
   rule_trung_left = 12,
   rule_trung_right = 13
};





class discreta_base;



// classes derived from base:

class integer;
   // self contains the integer value as a C (long)integer (int)

class longinteger;
   // self is a pointer to LONGINTEGER_REPRESENTATION
   // which contains the sign, the length
   // and a C array of chars containing
   // the decimal representation of the signless longinteger value

class discreta_matrix;
   // self is a pointer obtained from
   // calloc_m_times_n_objects().
   // this means that we have an array of m * n + 2 objects,
   // self points to the m * n array of user entries
```

```
   // and at offset [-2] we have m (as an integer object),
   // at offset [-1] we have n (as an integer object).
   // matrix access (via s_ij or via operator[])
   // is range checked.

class Vector;
   // self is a pointer obtained from
   // calloc_nobjects_plus_length().
   // this means that we have an array of n + 1 objects,
   // self points to the n array of user entries
   // and at offset [-1] we have the length l (as an integer object),
   // vector access (via s_i or via operator[])
   // is range checked.

class memory;
   // self is a pointer to char which has some additional
   // information stored at offset -3, -2, -1 in int4s.
   // these are alloc_length, used_length and cur_pointer.

class hollerith;
// there are so many string classes around so that I call
// my string class hollerith class!
// n.b.: Herman Hollerith (Buffalo 1860 - Washington 1929),
// American engineer; he invented
// statistical machines working with perforated cards
// In 1896, he founded the Tabulating Machine Corporation
// which later became IBM.


// classes derived from vector:

   class permutation;
      // a vector holding the images of the
      // points 0, 1, ..., l-1 under the permutation.
      // Note that the images are in 0, 1, ... , l-1 again!
      // the length is already stored in the vector.

   class number_partition;
      // a vector of length 2:
      // offset 0: the type (PARTITION_TYPE_VECTOR
      //                     or PARTITION_TYPE_EXPONENT)
      // offset 1: the self part holding the parts

   class unipoly; // derived from vector
   class bt_key; // derived from vector
   class database; // derived from vector
   class btree; // derived from vector
```

```
   class design_parameter_source; // derived from vector
   class design_parameter; // derived from vector


// auxiliary class, for the operator M[i][j] matrix access:

class matrix_access;





class domain;
class with;
class printing_mode;



// in global.cpp:

extern const char *discreta_home;
extern const char *discreta_arch;



void discreta_init();
discreta_base *callocobject(kind k);
void freeobject(discreta_base *p);
discreta_base *calloc_nobjects(int n, kind k);
void free_nobjects(discreta_base *p, int n);
discreta_base *calloc_nobjects_plus_length(int n, kind k);
void free_nobjects_plus_length(discreta_base *p);
discreta_base *calloc_m_times_n_objects(int m, int n, kind k);
void free_m_times_n_objects(discreta_base *p);
void printobjectkind(std::ostream& ost, kind k);
const char *kind_ascii(kind k);
const char *action_kind_ascii(action_kind k);
void uint4_swap(uint_4& x, uint_4& y);

std::ostream& operator<<(std::ostream& ost, class discreta_base& p);

int invert_mod_integer(int i, int p);
int remainder_mod(int i, int n);
void factor_integer(int n, Vector& primes, Vector& exponents);
void discreta_print_factorization(
      Vector& primes, Vector& exponents, std::ostream &o);
```

```
void print_factorization_hollerith(
      Vector& primes, Vector& exponents, hollerith &h);
int nb_primes(int n);
int factor_if_prime_power(int n, int *p, int *e);
int Euler(int n);
int Moebius(int i);
int NormRemainder(int a, int m);
int log2(int n);
int sqrt_mod(int a, int p, int verbose_level);
int sqrt_mod_involved(int a, int p, int verbose_level);
void html_head(std::ostream& ost,
      char *title_long, char *title_short);
void html_foot(std::ostream& ost);
void sieve(
      Vector &primes,
      int factorbase, int verbose_level);
void sieve_primes(
      Vector &v, int from, int to,
      int limit, int verbose_level);
void print_intvec_mod_10(Vector &v);
void stirling_second(
      int n, int k, int f_ordered,
      discreta_base &res, int verbose_level);
void stirling_first(
      int n, int k, int f_signless,
      discreta_base &res, int verbose_level);
void Catalan(int n, Vector &v, int verbose_level);
void Catalan_n(int n, Vector &v,
      discreta_base &res, int verbose_level);
void Catalan_nk_matrix(
      int n, discreta_matrix &Cnk,
      int verbose_level);
void Catalan_nk_star_matrix(
      int n, discreta_matrix &Cnk,
      int verbose_level);
void Catalan_nk_star(
      int n, int k, discreta_matrix &Cnk,
      discreta_base &res, int verbose_level);

void N_choose_K(
      discreta_base & n, int k, discreta_base & res);
void Binomial(
      int n, int k, discreta_base & n_choose_k);
void Krawtchouk(
      int n, int q, int i, int j, discreta_base & a);
// $\sum_{u=0}^{\min(i,j)} (-1)^u \cdot (q-1)^{i-u} \cdot {j \choose u} \cdot $
// ${n - j \choose i - u}$
```

```
//int ij2k(int i, int j, int n);
//void k2ij(int k, int & i, int & j, int n);
void tuple2_rank(
      int rank, int &i, int &j, int n, int f_injective);
int tuple2_unrank(
      int i, int j, int n, int f_injective);
void output_texable_string(std::ostream & ost, char *in);
void texable_string(char *in, char *out);
void the_first_n_primes(Vector &P, int n);
void midpoint_of_2(
      int *Px, int *Py, int i1, int i2, int idx);
void midpoint_of_5(
      int *Px, int *Py,
      int i1, int i2, int i3, int i4, int i5, int idx);
void ratio_int(
      int *Px, int *Py, int idx_from, int idx_to,
      int idx_result, double r);

enum printing_mode_enum current_printing_mode();
void call_system(char *cmd);
void fill_char(void *v, int cnt, int c);
int hash_int(int hash0, int a);
void queue_init(Vector &Q, int elt);
int queue_get_and_remove_first_element(Vector &Q);
int queue_length(Vector &Q);
void queue_append(Vector &Q, int elt);
void print_classification_tex(
      Vector &content, Vector &multiplicities);
void print_classification_tex(
      Vector &content, Vector &multiplicities,
      std::ostream& ost);
void perm2permutation(int *a, int n, permutation &p);
int Gauss_int(
      int *A, int f_special, int f_complete, int *base_cols,
   int f_P, int *P, int m, int n, int Pn,
   int q, int *add_table, int *mult_table,
   int *negate_table, int *inv_table, int verbose_level);
// returns the rank which is the number of entries in base_cols
void uchar_move(uchar *p, uchar *q, int len);
void int_vector_realloc(int *&p, int old_length, int new_length);
void int_vector_shorten(int *&p, int new_length);
void int_matrix_realloc(
      int *&p, int old_m, int new_m,
      int old_n, int new_n);
int code_is_irreducible(int k, int nmk, int idx_zero, int *M);
void fine_tune(
      layer1_foundations::field_theory::finite_field *F,
```

```
      int *mtxD, int verbose_level);



// internal representations:

typedef struct longinteger_representation LONGINTEGER_REPRESENTATION;
//typedef struct bitmatrix_representation BITMATRIX_REPRESENTATION;

//! DISCRETA internal class


typedef union {
   long int integer_value;
   char *char_pointer;
   int *int_pointer;
   discreta_base *vector_pointer;
   discreta_base *matrix_pointer;
   LONGINTEGER_REPRESENTATION *longinteger_rep;
} OBJECTSELF;

//! DISCRETA internal class to represent long integers


struct longinteger_representation {
   int sign;
   int len;
   char p[1];
};





//! DISCRETA base class. All DISCRETA classes are derived from this class




class discreta_base
{
   private:

   public:

   kind k;
   OBJECTSELF self;
```

```
discreta_base();
discreta_base(const discreta_base& x);
   // copy constructor
discreta_base& operator = (const discreta_base &x);
   // copy assignment
virtual ~discreta_base();
void freeself_discreta_base();
void freeself();
void freeself_kind(kind k);
void clearself() { self.vector_pointer = NULL; }

integer& as_integer()
   { return *(integer *)this; }
longinteger& as_longinteger()
   { return *(longinteger *)this; }
Vector& as_vector()
   { return *(Vector *)this; }
permutation& as_permutation()
   { return *(permutation *)this; }

number_partition& as_number_partition()
   { return *(number_partition *)this; }
discreta_matrix& as_matrix()
   { return *(discreta_matrix *)this; }
unipoly& as_unipoly()
   { return *(unipoly *)this; }
memory& as_memory()
   { return *(memory *)this; }
hollerith& as_hollerith()
   { return *(hollerith *)this; }
bt_key& as_bt_key()
   { return *(bt_key *)this; }
database& as_database()
   { return *(database *)this; }
btree& as_btree()
   { return *(btree *)this; }
design_parameter_source& as_design_parameter_source()
   { return *(design_parameter_source *)this; }
design_parameter& as_design_parameter()
   { return *(design_parameter *)this; }

integer& change_to_integer()
   { freeself(); c_kind(INTEGER); return as_integer(); }
longinteger& change_to_longinteger()
   { freeself(); c_kind(LONGINTEGER); return as_longinteger(); }
Vector& change_to_vector()
```

```
   { freeself(); c_kind(VECTOR); return as_vector(); }
permutation& change_to_permutation()
   { freeself(); c_kind(PERMUTATION); return as_permutation(); }
number_partition& change_to_number_partition()
   { freeself(); c_kind(NUMBER_PARTITION);
      return as_number_partition(); }
discreta_matrix& change_to_matrix()
   { freeself(); c_kind(MATRIX); return as_matrix(); }
unipoly& change_to_unipoly()
   { freeself(); c_kind(UNIPOLY); return as_unipoly(); }
memory& change_to_memory()
   { freeself(); c_kind(MEMORY); return as_memory(); }
hollerith& change_to_hollerith()
   { freeself(); c_kind(HOLLERITH); return as_hollerith(); }
bt_key& change_to_bt_key()
   { freeself(); c_kind(BT_KEY); return as_bt_key(); }
database& change_to_database()
   { freeself(); c_kind(DATABASE); return as_database(); }
btree& change_to_btree()
   { freeself(); c_kind(BTREE); return as_btree(); }
design_parameter_source& change_to_design_parameter_source()
   { freeself(); c_kind(DESIGN_PARAMETER_SOURCE);
      return as_design_parameter_source(); }
design_parameter& change_to_design_parameter()
   { freeself(); c_kind(DESIGN_PARAMETER);
      return as_design_parameter(); }

void *operator new(size_t, void *p) { return p; }
void settype_base();

kind s_kind();
   // select kind of object
virtual kind s_virtual_kind();
void c_kind(kind k);
   // compute kind of object:
   // changes the object kind to class k
   // preserves the self part of the object
void swap(discreta_base &a);
void copyobject(discreta_base &x);
   // this := x
virtual void copyobject_to(discreta_base &x);
   // x := this

virtual std::ostream& print(std::ostream&);
   // all kinds of printing,
   // the current printing mode is determined
   // by the global variable printing_mode
```

```
void print_to_hollerith(hollerith& h);
std::ostream& println(std::ostream&);
   // print() and newline
std::ostream& printobjectkind(std::ostream&);
   // prints the type of the object
std::ostream& printobjectkindln(std::ostream&);

long int &s_i_i();
   // select_as_integer_i
void m_i_i(long int i);
   // make_as_integer_i

virtual int compare_with(discreta_base &a);
   // -1 iff this < a
   // 0 iff this = a
   // 1 iff this > a
int eq(discreta_base &a);
int neq(discreta_base &a);
int le(discreta_base &a);
int lt(discreta_base &a);
int ge(discreta_base &a);
int gt(discreta_base &a);
int is_even();
int is_odd();


// arithmetic functions:

// multiplicative group:
void mult(discreta_base &x, discreta_base &y);
   // this := x * y
void mult_mod(
      discreta_base &x, discreta_base &y,
      discreta_base &p);
virtual void mult_to(discreta_base &x, discreta_base &y);
   // y := this * x
int invert();
   // this := this^(-1)
   // returns TRUE if the object was invertible,
   // FALSE otherwise
int invert_mod(discreta_base &p);
virtual int invert_to(discreta_base &x);
void mult_apply(discreta_base &x);
   // this := this * x
discreta_base& operator *= (discreta_base &y)
   { mult_apply(y); return *this; }
discreta_base& power_int(int l);
```

```
   // this := this^l, l >= 0
discreta_base& power_int_mod(int l, discreta_base &p);
discreta_base& power_longinteger(longinteger &l);
discreta_base& power_longinteger_mod(
      longinteger &l, discreta_base &p);
discreta_base& commutator(discreta_base &x, discreta_base &y);
   // this := x^{-1} * y^{-1} * x * y
discreta_base& conjugate(discreta_base &x, discreta_base &y);
   // this := y^{-1} * x * y
discreta_base& divide_by(discreta_base& x);
discreta_base& divide_by_exact(discreta_base& x);
int order();
int order_mod(discreta_base &p);


// additive group:
void add(discreta_base &x, discreta_base &y);
   // this := x + y
void add_mod(discreta_base &x, discreta_base &y, discreta_base &p);
virtual void add_to(discreta_base &x, discreta_base &y);
   // y := this + x
void negate();
   // this := -this;
virtual void negate_to(discreta_base &x);
   // x := - this
void add_apply(discreta_base &x);
   // this := this + x
discreta_base& operator += (discreta_base &y)
   { add_apply(y); return *this; }


virtual void normalize(discreta_base &p);
virtual void zero();
   // this := 0
virtual void one();
   // this := 1
virtual void m_one();
   // this := -1
virtual void homo_z(int z);
   // this := z
virtual void inc();
   // this := this + 1
virtual void dec();
   // this := this - 1
virtual int is_zero();
   // TRUE iff this = 0
virtual int is_one();
```

```
      // TRUE iff this = 1
   virtual int is_m_one();
      // TRUE iff this = -1
   discreta_base& factorial(int z);
      // this := z!
   discreta_base& i_power_j(int i, int j);
      // this := i^j

   virtual int compare_with_euclidean(
         discreta_base &a);
      // -1 iff this < a
      // 0 iff this = a
      // 1 iff this > a
   virtual void integral_division(
         discreta_base &x,
         discreta_base &q, discreta_base &r,
         int verbose_level);
   void integral_division_exact(
         discreta_base &x, discreta_base &q);
   void integral_division_by_integer(
         int x, discreta_base &q, discreta_base &r);
   void integral_division_by_integer_exact(
         int x, discreta_base &q);
   void integral_division_by_integer_exact_apply(int x);
   int is_divisor(discreta_base &y);
   void modulo(discreta_base &p);
   void extended_gcd(
         discreta_base &n, discreta_base &u,
         discreta_base &v, discreta_base &g,
         int verbose_level);
   void write_memory(memory &m, int debug_depth);
   void read_memory(memory &m, int debug_depth);
   int calc_size_on_file();
   void pack(memory & M, int verbose_level, int debug_depth);
   void unpack(memory & M, int verbose_level, int debug_depth);
   void save_ascii(std::ostream & f);
   void load_ascii(std::istream & f);
   void save_file(const char *fname);
   void load_file(const char *fname);
};


//! DISCRETA class to serialize data structures


class memory: public discreta_base
{
```

```
   public:
   memory();
   memory(const discreta_base &x);
      // copy constructor
   memory& operator = (const discreta_base &x);
      // copy assignment
   void settype_memory();
   ~memory();
   void freeself_memory();
   kind s_virtual_kind();
   void copyobject_to(discreta_base &x);
   std::ostream& print(std::ostream& ost);
   int & alloc_length()
      { return self.int_pointer[-3]; }
   int & used_length()
      { return self.int_pointer[-2]; }
   int & cur_pointer()
      { return self.int_pointer[-1]; }

   char & s_i(int i)
      { return self.char_pointer[i]; };
   char & operator [] (int i)
      { return s_i(i); }

   void init(int length, char *d);
   void alloc(int length);
   void append(int length, char *d);
   void realloc(int new_length);
   void write_char(char c);
   void read_char(char *c);
   void write_int(int i);
   void read_int(int *i);
   void read_file(char *fname, int verbose_level);
   void write_file(char *fname, int verbose_level);
   int multiplicity_of_character(char c);
   void compress(int verbose_level);
   void decompress(int verbose_level);
   int csf();
   void write_mem(memory & M, int debug_depth);
   void read_mem(memory & M, int debug_depth);

};


//! DISCRETA string class
```

```
class hollerith: public discreta_base
{
   public:
   hollerith();
      // constructor, sets the hollerith_pointer to NULL
   hollerith(char *p);
   hollerith(const discreta_base& x);
      // copy constructor
   hollerith& operator = (const discreta_base &x);
      // copy assignment

   void *operator new(size_t, void *p)
      { return p; }
   void settype_hollerith();

   ~hollerith();
   void freeself_hollerith();
      // delete the matrix
   kind s_virtual_kind();
   void copyobject_to(discreta_base &x);

   std::ostream& print(std::ostream&);
   int compare_with(discreta_base &a);

   char * s_unchecked()
      { return self.char_pointer; }
   char * s()
      { if (self.char_pointer)
      return self.char_pointer; else return (char *) ""; }
   void init(const char *p);
   void append(const char *p);
   void append_i(int i);
   void write_mem(memory & m, int debug_depth);
   void read_mem(memory & m, int debug_depth);
   int csf();
   void chop_off_extension_if_present(char *ext);
   void get_extension_if_present(char *ext);
   void get_current_date();
};

//! DISCRETA integer class



class integer: public discreta_base
{
   public:
```

```
integer();
integer(char *p);
integer(long int i);
integer(const discreta_base& x);
   // copy constructor
integer& operator = (const discreta_base &x);
   // copy assignment
void *operator new(size_t, void *p)
   { return p; }
void settype_integer();

~integer();
void freeself_integer();
kind s_virtual_kind();
void copyobject_to(discreta_base &x);

std::ostream& print(std::ostream&);

integer& m_i(long int i); // make_integer
long int & s_i()
   { return self.integer_value; }; // select_integer

int compare_with(discreta_base &a);

void mult_to(discreta_base &x, discreta_base &y);
int invert_to(discreta_base &x);

void add_to(discreta_base &x, discreta_base &y);
void negate_to(discreta_base &x);

void normalize(discreta_base &p);
void zero();
void one();
void m_one();
void homo_z(int z);
void inc();
void dec();
int is_zero();
int is_one();
int is_m_one();

int compare_with_euclidean(discreta_base &a);
void integral_division(
      discreta_base &x,
      discreta_base &q, discreta_base &r,
      int verbose_level);
```

```
   void rand(int low, int high);
   int log2();
};

#define LONGINTEGER_PRINT_DOTS
#define LONGINTEGER_DIGITS_FOR_DOT 6



//! DISCRETA  class for integers of arbitrary magnitude




class longinteger: public discreta_base
{
   public:
   longinteger();
   longinteger(int a);
   longinteger(const char *s);
   longinteger(const discreta_base& x);
      // copy constructor
   longinteger& operator = (const discreta_base &x);
      // copy assignment
   void *operator new(size_t, void *p) { return p; }
   void settype_longinteger();

   ~longinteger();
   void freeself_longinteger();
   kind s_virtual_kind();
   void copyobject_to(discreta_base &x);

   std::ostream& print(std::ostream&);

   LONGINTEGER_REPRESENTATION *s_rep();
   int& s_sign();
   int& s_len();
   char& s_p(int i);
   void allocate(int sign, const char *p);
   void allocate_internal(int sign, int len, const char *p);
   void allocate_empty(int len);
   void normalize_representation();

   int compare_with(discreta_base &b);
   int compare_with_unsigned(longinteger &b);

   void mult_to(discreta_base &x, discreta_base &y);
```

```
   int invert_to(discreta_base &x);
   void add_to(discreta_base &x, discreta_base &y);
   void negate_to(discreta_base &x);

   void zero();
   void one();
   void m_one();
   void homo_z(int z);
   void inc();
   void dec();
   int is_zero();
   int is_one();
   int is_m_one();
   int is_even();
   int is_odd();

   int compare_with_euclidean(discreta_base &b);
   void integral_division(
         discreta_base &x,
         discreta_base &q, discreta_base &r,
         int verbose_level);
   void square_root_floor(discreta_base &x);
   longinteger& Mersenne(int n);
   longinteger& Fermat(int n);
   int s_i();
   int retract_to_integer_if_possible(integer &x);
   int modp(int p);
   int ny_p(int p);
   void divide_out_int(int d);

   int Lucas_test_Mersenne(int m, int verbose_level);
};


//! DISCRETA vector class for vectors of DISCRETA objects




class Vector: public discreta_base
{
   public:
   Vector();
      // constructor, sets the vector_pointer to NULL
   Vector(const discreta_base& x);
      // copy constructor
   Vector& operator = (const discreta_base &x);
```

```
   // copy assignment

void *operator new(size_t, void *p) { return p; }
void settype_vector();
~Vector();
void freeself_vector();
   // delete the vector
kind s_virtual_kind();
void copyobject_to(discreta_base &x);

std::ostream& Print(std::ostream&);
std::ostream& print(std::ostream&);
std::ostream& print_unformatted(std::ostream& ost);
std::ostream& print_intvec(std::ostream& ost);

discreta_base & s_i(int i);
   // select i-th vector element
long int& s_ii(int i)
   { return s_i(i).s_i_i(); }
   // select i-th vector element as integer
void m_ii(int i, long int a) { s_i(i).m_i_i(a); }
   // make i-th vector element as integer (set value)
discreta_base & operator [] (int i)
   { return s_i(i); }
int s_l();
   // select vector length,
   // length is 0 if vector_pointer is NULL
void m_l(int l);
   // make vector of length l
   // allocates the memory and sets the objects to type BASE
void m_l_n(int l);
   // make vector of length l of integers, initializes with 0
void m_l_e(int l);
   // make vector of length l of integers, initializes with 1
void m_l_x(int l, discreta_base &x);
   // allocates a vector of l copies of x
Vector& realloc(int l);
void mult_to(discreta_base &x, discreta_base &y);
void add_to(discreta_base &x, discreta_base &y);
void inc();
void dec();

int compare_with(discreta_base &a);
void append_vector(Vector &v);
Vector& append_integer(int a);
Vector& append(discreta_base &x);
Vector& insert_element(int i, discreta_base& x);
```

```
Vector& get_and_delete_element(int i, discreta_base& x);
Vector& delete_element(int i);
void get_first_and_remove(discreta_base & x);
bool insert_sorted(discreta_base& x);
   // inserts x into the sorted vector x.
   // if there are already occurrences of x, the new x is added
   // behind the x already there.
   // returns true if the element was already in the vector.
bool search(discreta_base& x, int *idx);
   // returns TRUE if the object x has been found.
   // idx contains the position where the object which
   // has been found lies.
   // if there are more than one element equal to x in the vector,
   // the last one will be found.
   // if the element has not been found, idx contains the position of
   // the next larger element.
   // This is the position to insert x if required.
Vector& sort();
void sort_with_fellow(Vector &fellow);
Vector& sort_with_logging(permutation& p);
   // the permutation p tells where the sorted elements
   // lay before, i.e. p[i] is the position of the
   // sorted element i in the unsorted vector.


void sum_of_all_entries(discreta_base &x);





void n_choose_k_first(int n, int k);
   // computes the lexicographically first k-subset of {0,...,n-1}
int n_choose_k_next(int n, int k);
   // computes the lexicographically next k-subset
   // returns FALSE if there is no further k-subset
   // example: n = 4, k = 2
   // first gives (0,1),
   // next gives (0,2), then (0,3), (1,2), (1,3), (2,3).
void first_lehmercode(int n)
   { m_l_n(n); }
   // first lehmercode = 0...0 (n times)
int next_lehmercode();
   // computes the next lehmercode,
   // returns FALSE iff there is no next lehmercode.
   // the last lehmercode is n-1,n-2,...,2,1,0
   // example: n = 3,
```

```
   // first_lehmercode gives
   // 0,0,0,0
   // next_lehmercode gives
   // 0, 1, 0
   // 1, 0, 0
   // 1, 1, 0
   // 2, 0, 0
   // 2, 1, 0
void lehmercode2perm(permutation& p);
void q_adic(int n, int q);
int q_adic_as_int(int q);
void mult_scalar(discreta_base& a);
void first_word(int n, int q);
int next_word(int q);
void first_regular_word(int n, int q);
int next_regular_word(int q);
int is_regular_word();

void apply_permutation(permutation &p);
void apply_permutation_to_elements(permutation &p);
void content(Vector & c, Vector & where);
void content_multiplicities_only(Vector & c, Vector & mult);

int hip();
int hip1();
void write_mem(memory & m, int debug_depth);
void read_mem(memory & m, int debug_depth);
int csf();

void conjugate(discreta_base & a);
void conjugate_with_inverse(discreta_base & a);
void replace(Vector &v);
void vector_of_vectors_replace(Vector &v);
void extract_subvector(Vector & v, int first, int len);

int hamming_weight();
void scalar_product(Vector &w, discreta_base & a);
void hadamard_product(Vector &w);
void intersect(Vector& b, Vector &c);
int vector_of_vectors_overall_length();
void first_divisor(Vector &exponents);
int next_divisor(Vector &exponents);
int next_non_trivial_divisor(Vector &exponents);
void multiply_out(Vector &primes, discreta_base &x);
int hash(int hash0);
int is_subset_of(Vector &w);
void concatenation(Vector &v1, Vector &v2);
```

```
   void print_word_nicely(std::ostream &ost,
         int f_generator_labels, Vector &generator_labels);
   void print_word_nicely2(std::ostream &ost);
   void print_word_nicely_with_generator_labels(
         std::ostream &ost, Vector &generator_labels);
   void vector_of_vectors_lengths(Vector &lengths);
   void get_element_orders(Vector &vec_of_orders);
};

void merge(Vector &v1, Vector &v2, Vector &v3);
void merge_with_fellows(Vector &v1, Vector &v1_fellow,
   Vector &v2, Vector &v2_fellow,
   Vector &v3, Vector &v3_fellow);
void merge_with_value(Vector &idx1, Vector &idx2, Vector &idx3,
   Vector &val1, Vector &val2, Vector &val3);
void intersection_of_vectors(Vector& V, Vector& v);


//! DISCRETA permutation class



class permutation: public Vector
{
   public:
   permutation();
      // constructor, sets the vector_pointer to NULL
   permutation(const discreta_base& x);
      // copy constructor
   permutation& operator = (const discreta_base &x);
      // copy assignment
   void *operator new(size_t, void *p) { return p; }
   void settype_permutation();
   kind s_virtual_kind();
   ~permutation();
   void freeself_permutation();
   void copyobject_to(discreta_base &x);
   std::ostream& print(std::ostream&);
   std::ostream& print_list(std::ostream& ost);
   std::ostream& print_cycle(std::ostream& ost);
   void sscan(const char *s, int verbose_level);
   void scan(std::istream & is, int verbose_level);

   void m_l(int l);
   long int& s_i(int i);
   long int& operator [] (int i)
      { return s_i(i); }
```

```
   void mult_to(discreta_base &x, discreta_base &y);
   int invert_to(discreta_base &x);
   void one();
   int is_one();
   int compare_with(discreta_base &a);

   void write_mem(memory & m, int debug_depth);
   void read_mem(memory & m, int debug_depth);
   int csf();
   void get_fixpoints(Vector &f);
   void induce_action_on_blocks(permutation & gg, Vector & B);
   void induce3(permutation & b);
   void induce2(permutation & b);
   void induce_on_2tuples(permutation & p, int f_injective);
   void add_n_fixpoints_in_front(permutation & b, int n);
   void add_n_fixpoints_at_end(permutation & b, int n);
   void add_fixpoint_in_front(permutation & b);
   void embed_at(permutation & b, int n, int at);
   void remove_fixpoint(permutation & b, int i);
   void join(permutation & a, permutation & b);
   void cartesian_product_action(permutation & a, permutation & b);
   void Add2Cycle(int i0, int i1);
   void Add3Cycle(int i0, int i1, int i2);
   void Add4Cycle(int i0, int i1, int i2, int i3);
   void Add5Cycle(int i0, int i1, int i2, int i3, int i4);
   void AddNCycle(int first, int len);

   // influence the behavior of printing of permutations:
   void set_print_type_integer_from_zero();
   void set_print_type_integer_from_one();
   void set_print_type_PG_1_q_element(domain *dom);

   void convert_digit(int i, hollerith &a);
   void cycle_type(Vector& type, int verbose_level);
   int nb_of_inversions(int verbose_level);
   int signum(int verbose_level);
   int is_even(int verbose_level);
   void cycles(Vector &cycles);
   void restrict_to_subset(permutation &q, int first, int len);
};

void signum_map(discreta_base & x, discreta_base &d);

//! DISCRETA utility class for matrix access
```

```
class matrix_access {
public:
   int i;
   discreta_matrix *p;
   discreta_base & operator [] (int j);
};

//! DISCRETA matrix class



class discreta_matrix: public discreta_base
{
   public:
   discreta_matrix();
      // constructor, sets the matrix_pointer to NULL
   discreta_matrix(const discreta_base& x);
      // copy constructor
   discreta_matrix& operator = (const discreta_base &x);
      // copy assignment

   void *operator new(size_t, void *p)
      { return p; }
   void settype_matrix();

   ~discreta_matrix();
   void freeself_matrix();
      // delete the matrix
   kind s_virtual_kind();
   void copyobject_to(discreta_base &x);

   std::ostream& print(std::ostream&);
   int compare_with(discreta_base &a);

   discreta_matrix& m_mn(int m, int n);
      // make matrix of format m times n
      // allocates the memory and sets the objects to type BASE
   discreta_matrix& m_mn_n(int m, int n);
   discreta_matrix& realloc(int m, int n);


   int s_m();
   int s_n();
   discreta_base & s_ij(int i, int j);
      // select (i,j)-th matrix element
```

```
long int& s_iji(int i, int j)
   { return s_ij(i, j).s_i_i(); }
void m_iji(int i, int j, int a)
   { s_ij(i, j).m_i_i(a); }
   // make (i,j)-th vector element as integer (set value)

matrix_access operator [] (int i)
   { matrix_access ma = { i, this };  return ma; }
   // overload access operator

void mult_to(discreta_base &x, discreta_base &y);
void matrix_mult_to(discreta_matrix &x, discreta_base &y);
void vector_mult_to(Vector &x, discreta_base &y);
void multiply_vector_from_left(Vector &x, Vector &y);
int invert_to(discreta_base &x);
void add_to(discreta_base &x, discreta_base &y);
void negate_to(discreta_base &x);
void one();
void zero();
int is_zero();
int is_one();


int Gauss(int f_special, int f_complete,
      Vector& base_cols, int f_P,
      discreta_matrix& P, int verbose_level);
int rank();
int get_kernel(
      Vector& base_cols, discreta_matrix& kernel);
discreta_matrix& transpose();
int Asup2Ainf();
int Ainf2Asup();
int Asup2Acover();
int Acover2nl(Vector& nl);

void Frobenius(unipoly& m, int p, int verbose_level);
void Berlekamp(unipoly& m, int p, int verbose_level);
void companion_matrix(unipoly& m, int verbose_level);

void elements_to_unipoly();
void minus_X_times_id();
void X_times_id_minus_self();
void smith_normal_form(
      discreta_matrix& P, discreta_matrix& Pv,
      discreta_matrix& Q, discreta_matrix& Qv,
      int verbose_level);
int smith_eliminate_column(
```

```
      discreta_matrix& P, discreta_matrix& Pv, int i,
      int verbose_level);
int smith_eliminate_row(
      discreta_matrix& Q, discreta_matrix& Qv, int i,
      int verbose_level);
void multiply_2by2_from_left(int i, int j,
   discreta_base& aii, discreta_base& aij,
   discreta_base& aji, discreta_base& ajj,
   int verbose_level);
void multiply_2by2_from_right(int i, int j,
   discreta_base& aii, discreta_base& aij,
   discreta_base& aji, discreta_base& ajj,
   int verbose_level);

void to_vector_of_rows(Vector& v);
void from_vector_of_rows(Vector& v);
void to_vector_of_columns(Vector& v);
void from_vector_of_columns(Vector& v);
void evaluate_at(discreta_base& x);
void KX_module_order_ideal(
      int i, unipoly& mue, int verbose_level);
void KX_module_apply(
      unipoly& p, Vector& v);
void KX_module_join(
      Vector& v1, unipoly& mue1,
   Vector& v2, unipoly& mue2, Vector& v3,
   unipoly& mue3, int verbose_level);
void KX_cyclic_module_generator(
      Vector& v, unipoly& mue, int verbose_level);
void KX_module_minpol(
      unipoly& p, unipoly& m,
      unipoly& mue, int verbose_level);

void binomial(
      int n_min, int n_max, int k_min, int k_max);
void stirling_second(
      int n_min, int n_max, int k_min, int k_max,
      int f_ordered);
void stirling_first(
      int n_min, int n_max, int k_min, int k_max,
      int f_signless);
void binomial(
      int n_min, int n_max, int k_min, int k_max,
      int f_inverse);
int hip();
int hip1();
void write_mem(memory & m, int debug_depth);
```

```
   void read_mem(memory & M, int debug_depth);
   int csf();

   void calc_theX(int & nb_X, int *&theX);
   void apply_perms(
         int f_row_perm, permutation &row_perm,
      int f_col_perm, permutation &col_perm);
   void apply_col_row_perm(permutation &p);
   void apply_row_col_perm(permutation &p);
   void incma_print_ascii_permuted_and_decomposed(
         std::ostream &ost, int f_tex,
      Vector & decomp, permutation & p);
   void print_decomposed(
         std::ostream &ost,
         Vector &row_decomp, Vector &col_decomp);
   void incma_print_ascii(
         std::ostream &ost, int f_tex,
      int f_row_decomp, Vector &row_decomp,
      int f_col_decomp, Vector &col_decomp);
   void incma_print_latex(
         std::ostream &f,
      int f_row_decomp, Vector &row_decomp,
      int f_col_decomp, Vector &col_decomp,
      int f_labelling_points, Vector &point_labels,
      int f_labelling_blocks, Vector &block_labels);
   void incma_print_latex2(std::ostream &f,
      int width, int width_10,
      int f_outline_thin, const char *unit_length,
      const char *thick_lines,
      const char *thin_lines, const char *geo_line_width,
      int f_row_decomp, Vector &row_decomp,
      int f_col_decomp, Vector &col_decomp,
      int f_labelling_points, Vector &point_labels,
      int f_labelling_blocks, Vector &block_labels);
   void calc_hash_key(
         int key_len, hollerith & hash_key, int verbose_level);
   int is_in_center();
   void power_mod(int r, integer &P, discreta_matrix &C);
   int proj_order_mod(integer &P);
   void determinant(discreta_base &d, int verbose_level);
   void det(discreta_base & d, int verbose_level);
   void det_modify_input_matrix(
         discreta_base & d, int verbose_level);
   void save_as_inc_file(char *fname);
   void save_as_inc(std::ofstream &f);
};
```

```
void determinant_map(discreta_base & x, discreta_base &d);
int nb_PG_lines(int n, int q);



//! DISCRETA class for polynomials in one variable



class unipoly: public Vector
{
   public:
   unipoly();
      // constructor, sets the vector_pointer to NULL
   unipoly(const discreta_base& x);
      // copy constructor
   unipoly& operator = (const discreta_base &x);
      // copy assignment
   void *operator new(size_t, void *p) { return p; }
   void settype_unipoly();
   kind s_virtual_kind();
   ~unipoly();
   void freeself_unipoly();
   void copyobject_to(discreta_base &x);
   std::ostream& print(std::ostream&);
   std::ostream& print_as_vector(std::ostream& ost);

   void m_l(int l);
   int degree();

   void mult_to(discreta_base &x, discreta_base &y);
   void add_to(discreta_base &x, discreta_base &y);
   void negate_to(discreta_base &x);
   void one();
   void zero();
   void x();
   void x_to_the_i(int i);
   int is_one();
   int is_zero();
   int compare_with_euclidean(discreta_base &a);
   void integral_division(
         discreta_base &x,
         discreta_base &q, discreta_base &r,
         int verbose_level);
   void derive();
   int is_squarefree(int verbose_level);
   int is_irreducible_GFp(int p, int verbose_level);
```

```
   int is_irreducible(int q, int verbose_level);
   int is_primitive(
         int m, int p, Vector& vp, int verbose_level);
   void numeric_polynomial(int n, int q);
   int polynomial_numeric(int q);
   void singer_candidate(int p, int f, int b, int a);
   void Singer(int p, int f, int verbose_level);
   void get_an_irreducible_polynomial(
         int f, int verbose_level);
   void evaluate_at(
         discreta_base& x, discreta_base& y);
   void largest_divisor_prime_to(
         unipoly& q, unipoly& r);
   void monic();
   void normal_base(int p,
         discreta_matrix& F, discreta_matrix& N,
         int verbose_level);
   int first_irreducible_polynomial(int p,
         unipoly& m, discreta_matrix& F,
         discreta_matrix& N, Vector &v,
         int verbose_level);
   int next_irreducible_polynomial(int p,
         unipoly& m, discreta_matrix& F,
         discreta_matrix& N, Vector &v,
         int verbose_level);
   void normalize(discreta_base &p);
   void Xnm1(int n);
   void Phi(int n, int f_v);
   void weight_enumerator_MDS_code(
         int n, int k, int q,
         int verbose_level);
   void charpoly(int q, int size, int *mtx, int verbose_level);

};




#define PARTITION_TYPE_VECTOR 0
#define PARTITION_TYPE_EXPONENT 1

//! DISCRETA class for partitions of an integer
```

```
class number_partition: public Vector
{
   public:
   number_partition();
      // constructor, sets the vector_pointer to NULL
   number_partition(int n);
   void allocate_number_partition();
   number_partition(const discreta_base& x);
      // copy constructor
   number_partition& operator = (const discreta_base &x);
      // copy assignment
   void *operator new(size_t, void *p)
      { return p; }
   void settype_number_partition();
   kind s_virtual_kind();
   ~number_partition();
   void freeself_number_partition();
   void copyobject_to(discreta_base &x);
   std::ostream& print(std::ostream&);

   long int & s_type()
      { return Vector::s_i(0).as_integer().s_i(); }
   Vector & s_self()
      { return Vector::s_i(1).as_vector(); }

   void m_l(int l)
      { s_self().m_l_n(l); }
   int s_l()
      { return s_self().s_l(); }
   long int & s_i(int i)
      { return s_self().s_ii(i); }
   long int & operator [] (int i)
      { return s_self().s_ii(i); }

   void first(int n);
   int next();
   int next_exponent();
   int next_vector();
   int first_into_k_parts(int n, int k);
   int next_into_k_parts(int n, int k);
   int first_into_at_most_k_parts(int n, int k);
   int next_into_at_most_k_parts(int n, int k);
   int nb_parts();
   void conjugate();
   void type(number_partition &q);
   void multinomial(discreta_base &res, int f_v);
   void multinomial_ordered(discreta_base &res, int f_v);
```

```
   int sum_of_decreased_parts();

};



//! DISCRETA class for influencing arithmetic operations




class domain {
   private:
      domain_type the_type;
      discreta_base the_prime;
      unipoly *the_factor_poly;
      domain *the_sub_domain;
      layer1_foundations::field_theory::finite_field *F;

   public:
      domain(int p);
      domain(layer1_foundations::field_theory::finite_field *F);
   domain(unipoly *factor_poly, domain *sub_domain);

   domain_type type();
   layer1_foundations::field_theory::finite_field *get_F();
   int order_int();
   int order_subfield_int();
   int characteristic();
   int is_Orbiter_finite_field_domain();
   unipoly *factor_poly();
   domain *sub_domain();
};


// domain.cpp:

int has_domain();
domain *get_current_domain();
int is_GFp_domain(domain *& d);
int is_GFq_domain(domain *& d);
int is_Orbiter_finite_field_domain(domain *& d);
int is_finite_field_domain(domain *& d);
int finite_field_domain_order_int(domain * d);
int finite_field_domain_characteristic(domain * d);
int finite_field_domain_primitive_root();
void finite_field_domain_base_over_subfield(Vector & b);
```

```
void push_domain(domain *d);
void pop_domain(domain *& d);
domain *allocate_finite_field_domain(int q, int verbose_level);
void free_finite_field_domain(domain *dom);




//! DISCRETA class related to class domain




class with {
   private:
   public:

   with(domain *dom);
   ~with();
};


//! DISCRETA  class related to printing of objects



class printing_mode {
   private:
   public:
   printing_mode(enum printing_mode_enum printing_mode);
   ~printing_mode();
};




//! DISCRETA class for databases



class bt_key: public Vector
{
   public:
   bt_key();
      // constructor, sets the vector_pointer to NULL
   bt_key(const discreta_base& x);
```

```
      // copy constructor
   bt_key& operator = (const discreta_base &x);
      // copy assignment
   void *operator new(size_t, void *p) { return p; }
   void settype_bt_key();
   kind s_virtual_kind();
   ~bt_key();
   void freeself_bt_key();
   void copyobject_to(discreta_base &x);
   std::ostream& print(std::ostream&);

   enum bt_key_kind & type()
      { return (enum bt_key_kind&) Vector::s_i(0).as_integer().s_i(); }
   long int & output_size()
      { return Vector::s_i(1).as_integer().s_i(); }
   long int & int_vec_first()
      { return Vector::s_i(2).as_integer().s_i(); }
   long int & int_vec_len()
      { return Vector::s_i(3).as_integer().s_i(); }
   long int & field1()
      { return Vector::s_i(4).as_integer().s_i(); }
   long int & field2()
      { return Vector::s_i(5).as_integer().s_i(); }
   long int & f_ascending()
      { return Vector::s_i(6).as_integer().s_i(); }

   void init(
         enum bt_key_kind type, int output_size,
         long int field1, long int field2);
   void init_int8(long int field1, long int field2);
   void init_int4(long int field1, long int field2);
   void init_int2(long int field1, long int field2);
   void init_string(int output_size,
         long int field1, long int field2);
   void init_int8_vec(long int field1,
         long int field2, int vec_fst, int vec_len);
   void init_int4_vec(long int field1,
         long int field2, int vec_fst, int vec_len);
   void init_int2_vec(long int field1,
         long int field2, int vec_fst, int vec_len);
};

int bt_lexicographic_cmp(char *p1, char *p2);
int bt_key_int4_cmp(char *p1, char *p2);
int bt_key_int2_cmp(char *p1, char *p2);
void bt_key_print_int8(char **key, std::ostream& ost);
void bt_key_print_int4(char **key, std::ostream& ost);
```

```
void bt_key_print_int2(char **key, std::ostream& ost);
void bt_key_print(char *key, Vector& V, std::ostream& ost);
int bt_key_compare_int8(char **p_key1, char **p_key2);
int bt_key_compare_int4(char **p_key1, char **p_key2);
int bt_key_compare_int2(char **p_key1, char **p_key2);
int bt_key_compare(char *key1, char *key2, Vector& V, int depth);
void bt_key_fill_in_int8(char **p_key, discreta_base& key_op);
void bt_key_fill_in_int4(char **p_key, discreta_base& key_op);
void bt_key_fill_in_int2(char **p_key, discreta_base& key_op);
void bt_key_fill_in_string(
      char **p_key, int output_size, discreta_base& key_op);
void bt_key_fill_in(char *key, Vector& V, Vector& the_object);
void bt_key_get_int8(char **key, int_8 &i);
void bt_key_get_int4(char **key, int_4 &i);
void bt_key_get_int2(char **key, int_2 &i);

#define BTREEMAXKEYLEN 24
//#define BTREEMAXKEYLEN 48
//#define BTREEMAXKEYLEN 512


//! DISCRETA auxiliary class related to the class database


typedef struct keycarrier {
   char c[BTREEMAXKEYLEN];
} KEYCARRIER;

typedef KEYCARRIER KEYTYPE;

//! DISCRETA auxiliary class related to the class database


typedef struct datatype {
   uint_4 datref;
   uint_4 data_size;
} DATATYPE;

//#define DB_SIZEOF_HEADER 16
//#define DB_SIZEOF_HEADER_LOG 4
#define DB_POS_FILESIZE 4

#define DB_FILE_TYPE_STANDARD 1
#define DB_FILE_TYPE_COMPACT 2


//! DISCRETA class for a database
```

```
class database: public Vector
{
   public:
   database();
      // constructor, sets the vector_pointer to NULL
   database(const discreta_base& x);
      // copy constructor
   database& operator = (const discreta_base &x);
      // copy assignment
   void *operator new(size_t, void *p) { return p; }
   void settype_database();
   kind s_virtual_kind();
   ~database();
   void freeself_database();
   void copyobject_to(discreta_base &x);
   std::ostream& print(std::ostream&);

   Vector & btree_access()
      { return Vector::s_i(0).as_vector(); }
   btree & btree_access_i(int i)
      { return btree_access().s_i(i).as_btree(); }
   hollerith & filename()
      { return Vector::s_i(1).as_hollerith(); }
   long int & f_compress()
      { return Vector::s_i(2).as_integer().s_i(); }
   long int & objectkind()
      { return Vector::s_i(3).as_integer().s_i(); }
   long int & f_open()
      { return Vector::s_i(4).as_integer().s_i(); }
   long int & stream()
      { return Vector::s_i(5).as_integer().s_i(); }
   long int & file_size()
      { return Vector::s_i(6).as_integer().s_i(); }
   long int & file_type()
      { return Vector::s_i(7).as_integer().s_i(); }

   void init(
         const char *filename, int objectkind, int f_compress);
   void init_with_file_type(
         const char *filename,
      int objectkind, int f_compress, int file_type);

   void create(int verbose_level);
   void open(int verbose_level);
```

```
void close(int verbose_level);
void delete_files();
void put_file_size();
void get_file_size();
void user2total(int user, int &total, int &pad);
int size_of_header();
int size_of_header_log();

void add_object_return_datref(
      Vector &the_object,
      uint_4 &datref, int verbose_level);
void add_object(
      Vector &the_object, int verbose_level);
void delete_object(
      Vector& the_object, uint_4 datref, int verbose_level);
void get_object(
      uint_4 datref, Vector &the_object, int verbose_level);
void get_object(
      DATATYPE *data_type, Vector &the_object, int verbose_level);
void get_object_by_unique_int8(
      int btree_idx,
   int id, Vector& the_object, int verbose_level);
int get_object_by_unique_int8_if_there(
      int btree_idx,
   int id, Vector& the_object, int verbose_level);
long int get_highest_int8(
      int btree_idx);
void ith_object(
      int i, int btree_idx,
   Vector& the_object, int verbose_level);
void ith(
      int i, int btree_idx,
   KEYTYPE *key_type, DATATYPE *data_type,
   int verbose_level);
void print_by_btree(
      int btree_idx, std::ostream& ost);
void print_by_btree_with_datref(
      int btree_idx, std::ostream& ost);
void print_subset(
      Vector& datrefs, std::ostream& ost);
void extract_subset(
      Vector& datrefs,
   char *out_path, int verbose_level);
void search_int8(
      int btree_idx,
   long int imin, long int imax, Vector &datrefs,
   int verbose_level);
```

```
   void search_int8_2dimensional(
         int btree_idx0, long int imin0, long int imax0,
         int btree_idx1, long int imin1, long int imax1,
         Vector &datrefs, int verbose_level);
   void search_int8_multi_dimensional(Vector& btree_idx,
      Vector& i_min, Vector &i_max, Vector& datrefs,
      int verbose_level);

   int get_size_from_datref(
         uint_4 datref, int verbose_level);
   void add_data_DB(
         void *d,
      int size, uint_4 *datref, int verbose_level);
   void add_data_DB_standard(
         void *d,
      int size, uint_4 *datref, int verbose_level);
   void add_data_DB_compact(
         void *d,
      int size, uint_4 *datref, int verbose_level);
   void free_data_DB(
         uint_4 datref, int size, int verbose_level);

   void file_open(int verbose_level);
   void file_create(int verbose_level);
   void file_close(int verbose_level);
   void file_seek(int offset);
   void file_write(void *p, int size, int nb);
   void file_read(void *p, int size, int nb);
};


#define BTREEHALFPAGESIZE  128
#define BTREEMAXPAGESIZE (2 * BTREEHALFPAGESIZE)

#define BTREE_PAGE_LENGTH_LOG 7

/* Dateiformat:
 * In Block 0 sind AllocRec/NextFreeRec/RootRec gesetzt.
 * Block 1..AllocRec sind Datenpages.
 * Die freien Bloecke sind ueber NextFreeRec verkettet.
 * Der letzte freie Block hat NIL als Nachfolger.
 * Dateigroesse = (AllocRec + 1) * sizeof(PageTyp) */


//! DISCRETA auxiliary class related to the class database
```

```
typedef struct itemtyp {
   KEYTYPE Key;
   DATATYPE Data;
   int_4 Childs; // number of descendants through Ref
   int_4 Ref;
} ItemTyp;


//! DISCRETA auxiliary class related to the class database


typedef struct pagetyp {
   int_4 AllocRec;
   int_4 NextFreeRec;
   int_4 RootRec;

   int_4 NumItems;
   ItemTyp Item[BTREEMAXPAGESIZE + 1];
/* Item[0]           enthaelt keine Daten,
 *                   nur Ref/Childs ist verwendet.
 * Item[1..NumItems] fuer Daten und
 *                   Ref/Childs verwendet. */
} PageTyp;


//! DISCRETA auxiliary class related to the class database


typedef struct buffer {
   int_4 PageNum;
   int_4 unused;
   PageTyp Page;
   long align;
} Buffer;


//! DISCRETA class for a database



class btree: public Vector
{
   public:
   btree();
      // constructor, sets the vector_pointer to NULL
   btree(const discreta_base& x);
      // copy constructor
```

```
btree& operator = (const discreta_base &x);
   // copy assignment
void *operator new(size_t, void *p)
   { return p; }
void settype_btree();
kind s_virtual_kind();
~btree();
void freeself_btree();
void copyobject_to(discreta_base &x);
std::ostream& print(std::ostream&);

long int & f_duplicatekeys()
   { return Vector::s_i(0).as_integer().s_i(); }
Vector & key()
   { return Vector::s_i(1).as_vector(); }
hollerith & filename()
   { return Vector::s_i(2).as_hollerith(); }
long int & f_open()
   { return Vector::s_i(3).as_integer().s_i(); }
long int & stream()
   { return Vector::s_i(4).as_integer().s_i(); }
long int & buf_idx()
   { return Vector::s_i(5).as_integer().s_i(); }
long int & Root()
   { return Vector::s_i(6).as_integer().s_i(); }
long int & FreeRec()
   { return Vector::s_i(7).as_integer().s_i(); }
long int & AllocRec()
   { return Vector::s_i(8).as_integer().s_i(); }
long int & btree_idx()
   { return Vector::s_i(9).as_integer().s_i(); }
long int & page_table_idx()
   { return Vector::s_i(10).as_integer().s_i(); }

void init(
      const char *file_name, int f_duplicatekeys,
      int btree_idx);
void add_key_int4(int field1, int field2);
void add_key_int2(int field1, int field2);
void add_key_string(int output_size, int field1, int field2);
void key_fill_in(char *the_key, Vector& the_object);
void key_print(char *the_key, std::ostream& ost);

void create(int verbose_level);
void open(int verbose_level);
void close(int verbose_level);
```

```
void ReadInfo(int verbose_level);
void WriteInfo(int verbose_level);
int AllocateRec(int verbose_level);
void ReleaseRec(int x);
void LoadPage(Buffer *BF, int x, int verbose_level);
void SavePage(Buffer *BF, int verbose_level);

int search_string(discreta_base& key_op,
   int& pos, int verbose_level);
void search_interval_int8(long int i_min, long int i_max,
   int& first, int &len, int verbose_level);
void search_interval_int8_int8(long int l0, long int u0,
   long int l1, long int u1,
   int& first, int &len, int verbose_level);
void search_interval_int8_int8_int8(long int l0, long int u0,
   long int l1, long int u1, long int l2, long int u2,
   int& first, int &len, int verbose_level);
void search_interval_int8_int8_int8_int8(
      long int l0, long int u0,
      long int l1, long int u1,
      long int l2, long int u2,
      long int l3, long int u3,
      int& first, int &len, int verbose_level);
int search_int8_int8(
      long int data1, long int data2, int& idx,
      int verbose_level);
int search_unique_int8(long int i, int verbose_level);
int search_unique_int8_int8_int8_int8(long int i0, long int i1,
   long int i2, long int i3, int verbose_level);
   // returns -1 if an element whose key starts
   // with [i0,i1,i2,i3] could not be found or is not unique.
   // otherwise, the idx of that element is returned
int search_datref_of_unique_int8(long int i,
   int verbose_level);
int search_datref_of_unique_int8_if_there(long int i,
   int verbose_level);
long int get_highest_int8();
void get_datrefs(int first,
   int len, Vector& datrefs);

int search(void *pSearchKey,
   DATATYPE *pData, int *idx, int key_depth,
   int verbose_level);
int SearchBtree(int page,
   void *pSearchKey, DATATYPE *pData,
   Buffer *Buf, int *idx, int key_depth,
   int verbose_level);
```

```
   int SearchPage(Buffer *buffer,
      void *pSearchKey, DATATYPE *pSearchData,
      int *cur, int *x, int key_depth,
      int verbose_level);

   int length(int verbose_level);
   void ith(int l,
      KEYTYPE *key, DATATYPE *data, int verbose_level);
   int page_i_th(int l,
      Buffer *buffer, int *cur, int *i,
      int verbose_level);

   void insert_key(KEYTYPE *pKey,
      DATATYPE *pData,
      int verbose_level);
   void Update(int Node, int *Rise,
      ItemTyp *RisenItem,
      int *RisenNeighbourChilds,
      int f_v);
   void Split(Buffer *BF,
      ItemTyp *Item, int x,
      int *RisenNeighbourChilds,
      int verbose_level);

   void delete_ith(int idx, int verbose_level);
   void Delete(
         int Node, int& Underflow, int verbose_level);
   void FindGreatest(int Node1,
      int& Underflow, Buffer *DKBF, int x,
      int verbose_level);
   void Compensate(int Precedent,
      int Node, int Path, int& Underflow,
      int verbose_level);

   void print_all(std::ostream& ost);
   void print_range(int first, int len, std::ostream& ost);
   void print_page(int x, std::ostream& ost);
   void page_print(Buffer *BF, std::ostream& ost);
   void item_print(ItemTyp *item, int i, std::ostream& ost);

   void file_open();
   void file_create();
   void file_close();
   void file_write(PageTyp *page, const char *message);
   void file_read(PageTyp *page, const char *message);
   void file_seek(int page_no);
};
```

```
#define MAX_FSTREAM_TABLE 1000


extern int fstream_table_used[MAX_FSTREAM_TABLE];
extern std::fstream *fstream_table[MAX_FSTREAM_TABLE];

int fstream_table_get_free_entry();
void database_init(int verbose_level);
void database_exit(void);
int root_buf_alloc(void);
void root_buf_free(int i);



// ##############################################################################
// class page_table
// ##############################################################################



typedef struct btree_page_registry_key_pair btree_page_registry_key_pair;

//! DISCRETA internal class related to class database

struct btree_page_registry_key_pair {
   int x;
   int idx;
   int ref;
};


//! DISCRETA internal class related to class database

typedef class page_table page_table;

typedef page_table *ppage_table;


//! DISCRETA class for bulk storage



class page_table {
public:
   layer1_foundations::data_structures::page_storage *btree_pages;
   int btree_page_registry_length;
```

```
   int btree_page_registry_allocated_length;
   btree_page_registry_key_pair *btree_table;


   page_table();
   ~page_table();
   void init(int verbose_level);
   void reallocate_table(int verbose_level);
   void print();
   int search(int len, int btree_idx, int btree_x, int &idx);
   int search_key_pair(int len, btree_page_registry_key_pair *K, int &idx);
   void save_page(Buffer *BF, int buf_idx, int verbose_level);
   int load_page(Buffer *BF, int x, int buf_idx, int verbose_level);
   void allocate_rec(Buffer *BF, int buf_idx, int x, int verbose_level);
   void write_pages_to_file(btree *B, int buf_idx, int verbose_level);
};




void page_table_init(int verbose_level);
void page_table_exit(int verbose_level);
int page_table_alloc(int verbose_level);
void page_table_free(int idx, int verbose_level);
page_table *page_table_pointer(int slot);




//! DISCRETA class for the design parameters database




class design_parameter_source: public Vector
{
   public:
   design_parameter_source();
      // constructor, sets the Vector_pointer to NULL
   design_parameter_source(const discreta_base& x);
      // copy constructor
   design_parameter_source& operator = (const discreta_base &x);
      // copy assignment
   void *operator new(size_t, void *p) { return p; }
   void settype_design_parameter_source();
   kind s_virtual_kind();
   ~design_parameter_source();
```

```
   void freeself_design_parameter_source();
   void copyobject_to(discreta_base &x);
   std::ostream& print(std::ostream&);
   void print2(design_parameter& p, std::ostream& ost);

   long int & prev()
      { return Vector::s_i(0).as_integer().s_i(); }
   long int & rule()
      { return Vector::s_i(1).as_integer().s_i(); }
   hollerith & comment()
      { return Vector::s_i(2).as_hollerith(); }
   Vector & references()
      { return Vector::s_i(3).as_vector(); }
   hollerith & references_i(int i)
      { return references().s_i(i).as_hollerith(); }

   void init();
   void text(hollerith& h);
   void text2(design_parameter& p, hollerith& h);
   void text012(hollerith& s0, hollerith& s1, hollerith& s2);
   void text012_extended(
         design_parameter& p, hollerith& s0,
         hollerith& s1, hollerith& s2);
};

// design.cpp:
int design_parameters_admissible(
      int v, int t, int k, discreta_base &lambda);
int calc_delta_lambda(
      int v, int t, int k, int f_v);
void design_lambda_max(
      int t, int v, int k, discreta_base & lambda_max);
void design_lambda_max_half(
      int t, int v, int k, discreta_base & lambda_max_half);
void design_lambda_ijs_matrix(
      int t, int v, int k, discreta_base& lambda,
      int s, discreta_matrix & M);
void design_lambda_ijs(
      int t, int v, int k,
      discreta_base& lambda, int s, int i, int j,
      discreta_base & lambda_ijs);
void design_lambda_ij(
      int t, int v, int k,
      discreta_base& lambda, int i, int j,
      discreta_base & lambda_ij);
int is_trivial_clan(
      int t, int v, int k);
```

```
void print_clan_tex_int(
      int t, int v, int k);
void print_clan_tex_int(
      int t, int v, int k, int delta_lambda,
      discreta_base &m_max);
void print_clan_tex(
      discreta_base &t, discreta_base &v, discreta_base &k,
      int delta_lambda, discreta_base &m_max);
int is_ancestor(
      int t, int v, int k);
int is_ancestor(
      int t, int v, int k, int delta_lambda);
int calc_redinv(
      int t, int v, int k, int delta_lambda,
      int &c, int &T, int &V, int &K, int &Delta_lambda);
int calc_derinv(
      int t, int v, int k, int delta_lambda,
      int &c, int &T, int &V, int &K, int &Delta_lambda);
int calc_resinv(
      int t, int v, int k, int delta_lambda,
      int &c, int &T, int &V, int &K, int &Delta_lambda);
void design_mendelsohn_coefficient_matrix(
      int t, int m, discreta_matrix & M);
void design_mendelsohn_rhs(
      int v, int t, int k, discreta_base& lambda,
      int m, int s, Vector & rhs);
int design_parameter_database_already_there(
      database &D, design_parameter &p, int& idx);
void design_parameter_database_add_if_new(
      database &D, design_parameter &p,
      long int& highest_id, int verbose_level);
void design_parameter_database_closure(
      database &D, int highest_id_already_closed,
      int minimal_t, int verbose_level);
void design_parameter_database_read_design_txt(
      char *fname_design_txt, char *path_db,
      int f_form_closure, int minimal_t, int verbose_level);
void design_parameter_database_export_tex(
      char *path_db);
int determine_restricted_number_of_designs_t(
      database &D, btree &B,
   int btree_idx_tvkl, long int t, int first, int len);
int determine_restricted_number_of_designs_t_v(
      database &D, btree &B,
   int btree_idx_tvkl, long int t, long int v, int first, int len);
void prepare_design_parameters_from_id(
      database &D, long int id, hollerith& h);
```

```
void prepare_link(
      hollerith& link, int id);
void design_parameter_database_clans(
      char *path_db, int f_html, int verbose_level);
void design_parameter_database_family_report(
      char *path_db, int t, int v, int k,
      int lambda, int minimal_t);
void design_parameter_database_clan_report(
      char *path_db, Vector &ancestor, Vector &clan_lambda,
      Vector & clan_member, Vector & clan_member_path);
int Maxfit(int i, int j);


//! DISCRETA class for design parameters



class design_parameter: public Vector
{
   public:
   design_parameter();
      // constructor, sets the vector_pointer to NULL
   design_parameter(const discreta_base& x);
      // copy constructor
   design_parameter& operator = (const discreta_base &x);
      // copy assignment
   void *operator new(size_t, void *p)
      { return p; }
   void settype_design_parameter();
   kind s_virtual_kind();
   ~design_parameter();
   void freeself_design_parameter();
   void copyobject_to(discreta_base &x);
   std::ostream& print(std::ostream&);

   long int & id()
      { return Vector::s_i(0).as_integer().s_i(); }
   long int & t()
      { return Vector::s_i(1).as_integer().s_i(); }
   long int & v()
      { return Vector::s_i(2).as_integer().s_i(); }
   long int & K()
      { return Vector::s_i(3).as_integer().s_i(); }
   discreta_base & lambda()
      { return Vector::s_i(4); }
   Vector & source()
      { return Vector::s_i(5).as_vector(); }
```

```
   design_parameter_source & source_i(int i)
      { return source().s_i(i).as_design_parameter_source(); }

   void init();
   void init(int t, int v, int k, int lambda);
   void init(int t, int v, int k, discreta_base& lambda);
   void text(hollerith& h);
   void text_parameter(hollerith& h);
   void reduced_t(design_parameter& p);
   int increased_t(design_parameter& p);
   void supplementary_reduced_t(design_parameter& p);
   void derived(design_parameter& p);
   int derived_inverse(design_parameter& p);
   void supplementary_derived(design_parameter& p);
   void residual(design_parameter& p);
   void ancestor(
         design_parameter& p, Vector & path,
         int verbose_level);
   void supplementary_residual(design_parameter& p);
   int residual_inverse(design_parameter& p);
   int trung_complementary(design_parameter& p);
   int trung_left_partner(
         int& t1, int& v1, int& k1, discreta_base& lambda1,
      int& t_new, int& v_new, int& k_new, discreta_base& lambda_new);
   int trung_right_partner(
         int& t1, int& v1, int& k1, discreta_base& lambda1,
      int& t_new, int& v_new, int& k_new, discreta_base& lambda_new);
   int alltop(design_parameter& p);
   void complementary(design_parameter& p);
   void supplementary(design_parameter& p);
   int is_selfsupplementary();
   void lambda_of_supplementary(discreta_base& lambda_supplementary);

   void init_database(database& D, char *path);
};



// discreta_global.cpp:
void free_global_data();
void the_end(int t0);
void the_end_quietly(int t0);


}}}
```

```
#endif /* ORBITER_SRC_LIB_DISCRETA_DISCRETA_H_ */
```

Chapter 5

Layer 3 – Group Actions

5.1 Main Header File

```
// group_actions.h
//
// Anton Betten
//
// started:  August 13, 2005



#ifndef ORBITER_SRC_LIB_GROUP_ACTIONS_GROUP_ACTIONS_H_
#define ORBITER_SRC_LIB_GROUP_ACTIONS_GROUP_ACTIONS_H_



using namespace orbiter::layer1_foundations;


namespace orbiter {


//! groups and group actions, induced group actions

namespace layer3_group_actions {


//! a specific group action

namespace actions {

   // actions
   class action;
   class action_global;
```

```
   class action_pointer_table;
   class nauty_interface_with_group;
   class known_groups;
   class stabilizer_chain_base_data;

}

//! data structures for groups and group actions.

namespace data_structures_groups {

   // data_structures
   class group_container;
   class incidence_structure_with_group;
   class orbit_rep;
   class orbit_transversal;
   class orbit_type_repository;
   class schreier_vector_handler;
   class schreier_vector;
   class set_and_stabilizer;
   class translation_plane_via_andre_model;
   class union_find_on_k_subsets;
   class union_find;
   class vector_ge_description;
   class vector_ge;
   typedef class vector_ge *p_vector_ge;

}

//! an implementation of various types of permutation groups using stabilizer cha
ins

namespace groups {


   // groups
   class direct_product;
   class exceptional_isomorphism_O4;
   class linear_group_description;
   class linear_group;
   class matrix_group;
   class orbits_on_something;
   class permutation_group_create;
   class permutation_group_description;
   class permutation_representation_domain;
   class permutation_representation;
   class schreier;
```

```
   class schreier_sims;
   class sims;
   class strong_generators;
   class subgroup;
   class sylow_structure;
   class wreath_product;

   typedef class sims *p_sims;
   typedef sims *psims;
   typedef strong_generators *pstrong_generators;
   typedef class subgroup *psubgroup;

}

//! various kinds of induced group actions

namespace induced_actions {


   // induced_actions
   class action_by_conjugation;
   class action_by_representation;
   class action_by_restriction;
   class action_by_right_multiplication;
   class action_by_subfield_structure;
   class action_on_andre;
   class action_on_bricks;
   class action_on_cosets;
   class action_on_determinant;
   class action_on_factor_space;
   class action_on_flags;
   class action_on_galois_group;
   class action_on_grassmannian;
   class action_on_homogeneous_polynomials;
   class action_on_interior_direct_product;
   class action_on_k_subsets;
   class action_on_orbits;
   class action_on_orthogonal;
   class action_on_set_partitions;
   class action_on_sets;
   class action_on_sign;
   class action_on_spread_set;
   class action_on_subgroups;
   class action_on_wedge_product;
   class product_action;

}
```

```
//! interfaces to outside software such as gap, nauty, magma

namespace interfaces {

   class l3_interface_gap;
   class magma_interface;
   class nauty_interface_with_group;

}



//! enumeration to distinguish between the various types of group actions

enum symmetry_group_type {
   unknown_symmetry_group_t,
   matrix_group_t,
   perm_group_t,
   wreath_product_t,
   direct_product_t,
   permutation_representation_t,
   action_on_sets_t,
   action_on_subgroups_t,
   action_on_k_subsets_t,
   action_on_pairs_t,
   action_on_ordered_pairs_t,
   base_change_t,
   product_action_t,
   action_by_right_multiplication_t,
   action_by_restriction_t,
   action_by_conjugation_t,
   action_on_determinant_t,
   action_on_galois_group_t,
   action_on_sign_t,
   action_on_grassmannian_t,
   action_on_spread_set_t,
   action_on_orthogonal_t,
   action_on_cosets_t,
   action_on_factor_space_t,
   action_on_wedge_product_t,
   action_by_representation_t,
   action_by_subfield_structure_t,
   action_on_bricks_t,
   action_on_andre_t,
   action_on_orbits_t,
   action_on_flags_t,
```

```
   action_on_homogeneous_polynomials_t,
   action_on_set_partitions_t,
   action_on_interior_direct_product_t
};

//! enumeration specific to action_by_representation

enum representation_type {
   representation_type_nothing,
   representation_type_PSL2_on_conic
};

//! the strategy which is employed to create shallow Schreier trees

enum shallow_schreier_tree_strategy {
   shallow_schreier_tree_standard,
   shallow_schreier_tree_Seress_deterministic,
   shallow_schreier_tree_Seress_randomized,
   shallow_schreier_tree_Sajeeb
};




//! to distinguish between the various types of permutation groups

enum permutation_group_type {
   unknown_permutation_group_t,
   symmetric_group_t,
   cyclic_group_t,
   identity_group_t,
   dihedral_group_t,
   bsgs_t,
};


//! interface for the various types of group actions

union symmetry_group {
   groups::matrix_group *matrix_grp;
   groups::permutation_representation_domain *perm_grp;
   groups::wreath_product *wreath_product_group;
   groups::direct_product *direct_product_group;
   groups::permutation_representation *Permutation_representation;
   induced_actions::action_on_sets *on_sets;
   induced_actions::action_on_subgroups *on_subgroups;
   induced_actions::action_on_k_subsets *on_k_subsets;
```

```
   induced_actions::product_action *product_action_data;
   induced_actions::action_by_right_multiplication *ABRM;
   induced_actions::action_by_restriction *ABR;
   induced_actions::action_by_conjugation *ABC;
   induced_actions::action_on_determinant *AD;
   induced_actions::action_on_galois_group *on_Galois_group;
   induced_actions::action_on_sign *OnSign;
   induced_actions::action_on_grassmannian *AG;
   induced_actions::action_on_spread_set *AS;
   induced_actions::action_on_orthogonal *AO;
   induced_actions::action_on_cosets *OnCosets;
   induced_actions::action_on_factor_space *AF;
   induced_actions::action_on_wedge_product *AW;
   induced_actions::action_by_representation *Rep;
   induced_actions::action_by_subfield_structure *SubfieldStructure;
   induced_actions::action_on_bricks *OnBricks;
   induced_actions::action_on_andre *OnAndre;
   induced_actions::action_on_orbits *OnOrbits;
   induced_actions::action_on_flags *OnFlags;
   induced_actions::action_on_homogeneous_polynomials *OnHP;
   induced_actions::action_on_set_partitions *OnSetPartitions;
   induced_actions::action_on_interior_direct_product *OnInteriorDirectProduct;
};




}}



#include "./actions/actions.h"
#include "data_structures/l3_data_structures.h"
#include "./groups/groups.h"
#include "./induced_actions/induced_actions.h"
#include "interfaces/l3_interfaces.h"



#endif /* ORBITER_SRC_LIB_GROUP_ACTIONS_GROUP_ACTIONS_H_ */
```

5.2 Actions

```
// group_actions.h
//
// Anton Betten
//
// moved here from action.h: July 28, 2018
// based on action.h which was started:  August 13, 2005



#ifndef ORBITER_SRC_LIB_GROUP_ACTIONS_ACTIONS_ACTIONS_H_
#define ORBITER_SRC_LIB_GROUP_ACTIONS_ACTIONS_ACTIONS_H_



namespace orbiter {

namespace layer3_group_actions {


namespace actions {


// ##############################################################################
// action.cpp
// ##############################################################################

//! a permutation group in a fixed action.
/*! The class action provides a unified interface to a permutation group.
 * A permutation group in Orbiter always acts on the set {0,...,degree-1},
 * where degree is the degree stored in the action class.
 * The primary goal of this class is to provide the functionality for
 * group actions. Many different types of group actions are supported.
 * The way in which a group action is realized is by means of the following
 * two components:
 *
 * symmetry_group G
 * symmetry_group_type type_G
 *
 * The type is an enumeration type which specifies the type of group action.
 * The component G is a union consisting of all possible pointer types
 * to objects of action type. Depending on the type of group action, type_G
 * is set and G holds a pointer to the specific class implementing this type
 * of action. There are the atomic types of group action and there are the
 * induced action types. At present, there are exactly two atomic types.
 * One is matrix_group, which represents a matrix group over a finite field.
 * The other one is perm_group, which represents a abstract permutation group.
```

```
* An abstract permutation group is a group where the elements are given
* as list of images in the form of a vector.
* We distinguish between matrix groups and abstract permutation group because
* the elements of a matrix group are given by matrices (possibly plus
* a vector, and possibly plus a field automorphism). The elements
* of a matrix group are not stored as permutations.
* Because of this, a matrix group can be more efficient than the corresponding
* isomorphic abstract permutation group. For instance, group multiplication
* for matrix group elements is matrix multiplication. Group multiplication
* for abstract permutation groups is composition of the list of images.
* Matrix group multiplication seems faster than composing lists of images,
* at least asymptotically.
*
* Optionally, the class action also serves as a means to represent a group by
* means of a stabilizer chain (sims chain). This proves to be a bit tricky
* because the class sims (implementing a sims chain) requires an action object
* to get going. This is a kind of chicken-and-egg problem. The action needs
* a sims, and the sims needs an action. The solution is to have a bit
* of replication of code. The action class has its own tiny implementation of
* a stabilizer chain. This allows a group to be set up in action without the
* need for a sims object. It is helpful that we know a stabilizer chain
* for the projective linear groups, so we can set up the tiny version
* of a sims chain in action without using a sims object. The
* problem of setting up a group arises within the various init
* function in the action class. The functionality for known stabilizer chains
* is pushed down in the foundations library, in order to keep the
* mathematics for stabilizer chains away from the implementation of the
* stabilizer chains and action classes. The action of the projective group
* on the set of points of projective space is called the natural action.
*
*
* In most cases of induced actions, it is not a good idea to
* replace the natural action by the induced action. The problem with
* actions for the purposes of representing groups as sims chains
* is the following. A sims chain is efficient only if the maximal
* degree of the basic actions (the actions in the stabilizer chain)
* is small. For the basic action of projective groups, this condition
* is usually met. For most induced actions, the maximal degree is
* large because the action has an even larger degree than the
* original basic action. For this reason, it is not advisable to represent
* a group in any of the induced actions. This means that we will carry
* two actions around at the same time. The first action is the basic action
* which represents the group in the natural action. The second action is
* the induced action which we desire in our particular application.
* This action will be use for the purposes of the group action only.
* It will not be used for a stabilizer chain for the group.
* The poset classification algorithm takes two actions as input.
```

```
 * The first action will be used to represent groups or subgroups.
 * The second action is the action on the poset of interest.
 *
 *
 */
class action {
public:


   /** the symmetry group is a permutation group
    *
    */
   int f_allocated;

   /** the type of group */
   symmetry_group_type type_G;

   /** a pointer to the implementation of the group.
    * symmetry_group is a union of pointer types.
    */
   symmetry_group G;


   /** Whether the group has a subaction.
    * For instance, induced actions have subactions. */
   int f_has_subaction;
   int f_subaction_is_allocated;

   /** the subaction */
   action *subaction;


   /** whether the group has strong generators */
   int f_has_strong_generators;

   /** strong generating set for the group */
   groups::strong_generators *Strong_gens;

   /** the size of the set we act on */
   long int degree;


   /** whether the action is linear (including semilinear) */
   int f_is_linear;
      // matrix_group_t,
      // action_on_wedge_product_t,
      // action_by_representation_t
```

```
/** the dimension if we are linear */
int dimension;


/** the number of int needed to store one group element */
int elt_size_in_int;

/** the number of char needed
 * to store a group element in the compressed form */
int coded_elt_size_in_char;


/** the number of int that are needed to
 * make an element of this group
 * using the make_element function */
int make_element_size;

/** the number of int that are needed to
 * represent a point in low-level format
 * (input and output in element_image_of_low_level
 * point to that many int) */
int low_level_point_size;

int f_has_sims;
/** sims chain for the group */
groups::sims *Sims;

int f_has_kernel;
/** kernel of the action */
groups::sims *Kernel;

int f_group_order_is_small;


int f_has_stabilizer_chain;

stabilizer_chain_base_data *Stabilizer_chain;

known_groups *Known_groups;


action_pointer_table *ptr;

/** temporary elements */
int *Elt1, *Elt2, *Elt3, *Elt4, *Elt5;
int *eltrk1, *eltrk2, *eltrk3, *elt_mult_apply;
```

```
uchar *elt1;
char *element_rw_memory_object;
   // [coded_elt_size_in_char]
   // for element_write_to_memory_object,
   // element_read_from_memory_object



/** a fancy label for the group */
std::string label;

/** a fancy label for the group for latex */
std::string label_tex;



// action.cpp
action();
~action();
void null();
void freeself();

int f_has_base();
int base_len();
void set_base_len(int base_len);
long int &base_i(int i);
long int *&get_base();
int &transversal_length_i(int i);
int *&get_transversal_length();
long int &orbit_ij(int i, int j);
long int &orbit_inv_ij(int i, int j);

void null_element_data();
void allocate_element_data();
void free_element_data();

int find_non_fixed_point(void *elt, int verbose_level);
int find_fixed_points(void *elt,
   int *fixed_points, int verbose_level);
int count_fixed_points(void *elt, int verbose_level);
int test_if_set_stabilizes(int *Elt,
   int size, long int *set, int verbose_level);
void map_a_set_based_on_hdl(long int *set,
      long int *image_set,
      int n, action *A_base, int hdl, int verbose_level);
void map_a_set(
      long int *set,
```

```
      long int *image_set,
   int n, int *Elt, int verbose_level);
void map_a_set_and_reorder(
      long int *set, long int *image_set,
   int n, int *Elt, int verbose_level);
void print_all_elements();

void init_sims_only(groups::sims *G, int verbose_level);
void compute_strong_generators_from_sims(int verbose_level);
void init_base_from_sims(groups::sims *G, int verbose_level);
int element_has_order_two(int *E1, int verbose_level);
int product_has_order_two(int *E1, int *E2, int verbose_level);
int product_has_order_three(int *E1, int *E2, int verbose_level);
int element_order(void *elt);
int element_order_and_cycle_type(void *elt, int *cycle_type);
int element_order_and_cycle_type_verbose(
      void *elt, int *cycle_type, int verbose_level);
int element_order_if_divisor_of(void *elt, int o);
void compute_all_point_orbits(groups::schreier &S,
      data_structures_groups::vector_ge &gens,
      int verbose_level);

/** the index of the first moved base point */
int depth_in_stab_chain(int *Elt);


   /** all strong generators that
    * leave base points 0,..., depth - 1 fix */
void strong_generators_at_depth(
      int depth,
      data_structures_groups::vector_ge &gen,
      int verbose_level);
void compute_point_stabilizer_chain(
      data_structures_groups::vector_ge &gen,
      groups::sims *S, int *sequence, int len,
   int verbose_level);
void compute_stabilizer_orbits(
      data_structures::partitionstack *&Staborbits,
      int verbose_level);
int check_if_in_set_stabilizer(
      int *Elt,
      int size, long int *set, int verbose_level);
int check_if_transporter_for_set(
      int *Elt,
      int size,
      long int *set1, long int *set2,
      int verbose_level);
```

```
void find_strong_generators_at_level(
      int base_len,
   long int *the_base, int level,
   data_structures_groups::vector_ge &gens,
   data_structures_groups::vector_ge &subset_of_gens,
   int verbose_level);
void make_element_from_permutation_representation(
      int *Elt,
      groups::sims *S, int *data, int verbose_level);
void make_element_from_base_image(
      int *Elt, groups::sims *S,
      int *data, int verbose_level);
void make_element_2x2(
      int *Elt, int a0, int a1, int a2, int a3);
void make_element_from_string(
      int *Elt,
      std::string &data_string, int verbose_level);
void make_element(
      int *Elt, int *data, int verbose_level);
void element_power_int_in_place(int *Elt,
   int n, int verbose_level);
void word_in_ab(int *Elt1, int *Elt2, int *Elt3,
   const char *word, int verbose_level);
void group_order(ring_theory::longinteger_object &go);
long int group_order_lint();
void element_print_base_images(int *Elt);
void element_print_base_images(
      int *Elt, std::ostream &ost);
void element_print_base_images_verbose(int *Elt,
      std::ostream &ost, int verbose_level);
void element_base_images(
      int *Elt, int *base_images);
void element_base_images_verbose(int *Elt,
   int *base_images, int verbose_level);
void minimize_base_images(
      int level, groups::sims *S,
   int *Elt, int verbose_level);
void get_generators_from_ascii_coding(
      std::string &ascii_coding,
      data_structures_groups::vector_ge *&gens,
      int *&tl, int verbose_level);
void lexorder_test(
      long int *set, int set_sz, int &set_sz_after_test,
      data_structures_groups::vector_ge *gens,
      int max_starter,
   int verbose_level);
void compute_orbits_on_points(
```

```
      groups::schreier *&Sch,
      data_structures_groups::vector_ge *gens,
      int verbose_level);

void point_stabilizer_any_point(
      int &pt,
      groups::schreier *&Sch, groups::sims *&Stab,
      groups::strong_generators *&stab_gens,
   int verbose_level);
void point_stabilizer_any_point_with_given_group(
      groups::strong_generators *input_gens,
   int &pt,
   groups::schreier *&Sch, groups::sims *&Stab,
   groups::strong_generators *&stab_gens,
   int verbose_level);
void make_element_which_moves_a_line_in_PG3q(
      geometry::grassmann *Gr,
   long int line_rk, int *Elt,
   int verbose_level);
int matrix_group_dimension();
field_theory::finite_field *matrix_group_finite_field();
int is_semilinear_matrix_group();
int is_projective();
int is_affine();
int is_general_linear();
int is_matrix_group();
groups::matrix_group *get_matrix_group();


// action_group_theory.cpp:
void report_groups_and_normalizers(
      std::ostream &ost,
      int nb_subgroups,
      groups::strong_generators *H_gens,
      groups::strong_generators *N_gens,
      int verbose_level);
void element_conjugate_bvab(int *Elt_A,
   int *Elt_B, int *Elt_C, int verbose_level);
void element_conjugate_babv(int *Elt_A,
   int *Elt_B, int *Elt_C, int verbose_level);
void element_commutator_abavbv(int *Elt_A,
   int *Elt_B, int *Elt_C, int verbose_level);
void compute_projectivity_subgroup(
      groups::strong_generators *&projectivity_gens,
      groups::strong_generators *Aut_gens,
      int verbose_level);
```

```
// action_indexing_cosets.cpp
void coset_unrank(groups::sims *G,
      groups::sims *U, long int rank,
   int *Elt, int verbose_level);
long int coset_rank(
      groups::sims *G, groups::sims *U,
   int *Elt, int verbose_level);
   // used in generator::coset_unrank and generator::coset_rank
   // which in turn are used by
   // generator::orbit_element_unrank and
   // generator::orbit_element_rank

// action_init.cpp

void init_group_from_generators(
      int *group_generator_data,
   int group_generator_size,
   int f_group_order_target,
   const char *group_order_target,
   data_structures_groups::vector_ge *gens,
   groups::strong_generators *&Strong_gens,
   int verbose_level);
void init_group_from_generators_by_base_images(
      groups::sims *S,
      int *group_generator_data, int group_generator_size,
      int f_group_order_target,
      const char *group_order_target,
      data_structures_groups::vector_ge *gens,
      groups::strong_generators *&Strong_gens_out,
      int verbose_level);
void build_up_automorphism_group_from_aut_data(
      int nb_auts,
   int *aut_data,
   groups::sims &S, int verbose_level);

groups::sims *create_sims_from_generators_with_target_group_order_factorized(
      data_structures_groups::vector_ge *gens,
      int *tl, int len, int verbose_level);
groups::sims *create_sims_from_generators_with_target_group_order_lint(
      data_structures_groups::vector_ge *gens,
      long int target_go, int verbose_level);
groups::sims *create_sims_from_generators_with_target_group_order(
      data_structures_groups::vector_ge *gens,
      ring_theory::longinteger_object &target_go,
   int verbose_level);
groups::sims *create_sims_from_generators_without_target_group_order(
```

```
      data_structures_groups::vector_ge *gens,
      int verbose_level);
groups::sims *create_sims_from_single_generator_without_target_group_order(
   int *Elt, int verbose_level);
groups::sims *create_sims_from_generators_randomized(
      data_structures_groups::vector_ge *gens,
      int f_target_go, ring_theory::longinteger_object &target_go,
   int verbose_level);
groups::sims *create_sims_for_centralizer_of_matrix(
      int *Mtx, int verbose_level);
void init_automorphism_group_from_group_table(
      std::string &fname_base,
   int *Table, int group_order, int *gens, int nb_gens,
   groups::strong_generators *&Aut_gens,
   int verbose_level);


// action_induce.cpp

action *induced_action_on_interior_direct_product(
      int nb_rows,
      int verbose_level);
action *induced_action_on_set_partitions(
      int partition_class_size,
      int verbose_level);

/** Create the induced action on lines in PG(n-1,q)
 * using an action_on_grassmannian object */
void init_action_on_lines(
      action *A, field_theory::finite_field *F,
   int n, int verbose_level);

void induced_action_by_representation_on_conic(
      action *A_old,
   int f_induce_action, groups::sims *old_G,
   int verbose_level);
void induced_action_on_cosets(
      induced_actions::action_on_cosets *A_on_cosets,
   int f_induce_action, groups::sims *old_G,
   int verbose_level);
void induced_action_on_factor_space(
      action *A_old,
      induced_actions::action_on_factor_space *AF,
   int f_induce_action, groups::sims *old_G,
   int verbose_level);
action *induced_action_on_grassmannian(
      int k,
```

```
   int verbose_level);
void induced_action_on_grassmannian(
      action *A_old,
      induced_actions::action_on_grassmannian *AG,
   int f_induce_action, groups::sims *old_G,
   int verbose_level);
void induced_action_on_spread_set(
      action *A_old,
      induced_actions::action_on_spread_set *AS,
   int f_induce_action, groups::sims *old_G,
   int verbose_level);
void induced_action_on_orthogonal(
      action *A_old,
      induced_actions::action_on_orthogonal *AO,
   int f_induce_action, groups::sims *old_G,
   int verbose_level);
action *induced_action_on_wedge_product(
      int verbose_level);
void induced_action_by_subfield_structure(
      action *A_old,
      induced_actions::action_by_subfield_structure
         *SubfieldStructure,
   int f_induce_action, groups::sims *old_G,
   int verbose_level);
void induced_action_on_determinant(
      groups::sims *old_G,
   int verbose_level);
void induced_action_on_Galois_group(
      groups::sims *old_G, int verbose_level);
void induced_action_on_sign(groups::sims *old_G,
   int verbose_level);
action *create_induced_action_by_conjugation(
      groups::sims *Base_group, int f_ownership,
      int verbose_level);
void induced_action_by_conjugation(
      groups::sims *old_G,
      groups::sims *Base_group, int f_ownership,
   int f_basis, int verbose_level);
void induced_action_by_right_multiplication(
   int f_basis, groups::sims *old_G,
   groups::sims *Base_group, int f_ownership,
   int verbose_level);
action *create_induced_action_on_sets(
      int nb_sets,
   int set_size, long int *sets,
   int verbose_level);
void induced_action_on_sets(
```

```
      action &old_action, groups::sims *old_G,
   int nb_sets, int set_size, long int *sets,
   int f_induce_action, int verbose_level);
action *create_induced_action_on_subgroups(
      groups::sims *S,
   int nb_subgroups, int group_order,
   groups::subgroup **Subgroups, int verbose_level);
void induced_action_on_subgroups(
      action *old_action,
      groups::sims *S,
   int nb_subgroups, int group_order,
   groups::subgroup **Subgroups,
   int verbose_level);
void induced_action_by_restriction_on_orbit_with_schreier_vector(
   action &old_action,
   int f_induce_action, groups::sims *old_G,
   data_structures_groups::schreier_vector *Schreier_vector,
   int pt, int verbose_level);
void original_point_labels(
      long int *points, int nb_points,
      long int *&original_points, int verbose_level);
action *restricted_action(
      long int *points, int nb_points,
   int verbose_level);
action *create_induced_action_by_restriction(
      groups::sims *S, int size,
      long int *set, int f_induce,
      int verbose_level);
void induced_action_by_restriction_internal_function(
      action &old_action,
   int f_induce_action, groups::sims *old_G,
   int nb_points, long int *points,
   int verbose_level);
   // uses action_by_restriction data type
void induced_action_on_pairs(
      action &old_action, groups::sims *old_G,
   int verbose_level);
action *create_induced_action_on_ordered_pairs(
      int verbose_level);
void induced_action_on_ordered_pairs(
      action &old_action,
      groups::sims *old_G,
   int verbose_level);
void induced_action_on_k_subsets(
      action &old_action, int k,
   int verbose_level);
void induced_action_on_orbits(
```

```
      action *old_action,
      groups::schreier *Sch, int f_play_it_safe,
   int verbose_level);
void induced_action_on_flags(
      action *old_action,
   int *type, int type_len,
   int verbose_level);
void induced_action_on_bricks(
      action &old_action,
   combinatorics::brick_domain *B, int f_linear_action,
   int verbose_level);
void induced_action_on_andre(
      action *An,
   action *An1,
   geometry::andre_construction *Andre,
   int verbose_level);
void setup_product_action(
      action *A1, action *A2,
   int f_use_projections, int verbose_level);
void induced_action_on_homogeneous_polynomials(
      action *A_old,
      ring_theory::homogeneous_polynomial_domain *HPD,
   int f_induce_action, groups::sims *old_G,
   int verbose_level);
void induced_action_on_homogeneous_polynomials_given_by_equations(
   action *A_old,
   ring_theory::homogeneous_polynomial_domain *HPD,
   int *Equations, int nb_equations,
   int f_induce_action, groups::sims *old_G,
   int verbose_level);
void induced_action_recycle_sims(
      action &old_action,
   int verbose_level);
void induced_action_override_sims(
      action &old_action,
      groups::sims *old_G,
   int verbose_level);
void induce(
      action *old_action, groups::sims *old_G,
   int base_of_choice_len,
   long int *base_of_choice,
   int verbose_level);
int least_moved_point_at_level(int level,
   int verbose_level);
void lex_least_base_in_place(int verbose_level);
void lex_least_base(
      action *old_action, int verbose_level);
```

```
int test_if_lex_least_base(int verbose_level);
void base_change_in_place(
      int size, long int *set, int verbose_level);
void base_change(
      action *old_action,
   int size, long int *set, int verbose_level);
void create_orbits_on_subset_using_restricted_action(
      action *&A_by_restriction,
      groups::schreier *&Orbits, groups::sims *S,
      int size, long int *set,
      int verbose_level);
void create_orbits_on_sets_using_action_on_sets(
      action *&A_on_sets,
      groups::schreier *&Orbits, groups::sims *S,
      int nb_sets, int set_size, long int *sets,
      int verbose_level);
int choose_next_base_point_default_method(
      int *Elt, int verbose_level);
void generators_to_strong_generators(
   int f_target_go,
   ring_theory::longinteger_object &target_go,
   data_structures_groups::vector_ge *gens,
   groups::strong_generators *&Strong_gens,
   int verbose_level);

// action_io.cpp:
void report(
      std::ostream &ost, int f_sims, groups::sims *S,
      int f_strong_gens, groups::strong_generators *SG,
      graphics::layered_graph_draw_options *LG_Draw_options,
      int verbose_level);
void report_what_we_act_on(
      std::ostream &ost,
      graphics::layered_graph_draw_options *O,
      int verbose_level);



void read_orbit_rep_and_candidates_from_files_and_process(
      std::string &prefix,
   int level, int orbit_at_level, int level_of_candidates_file,
   void (*early_test_func_callback)(long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      void *data, int verbose_level),
   void *early_test_func_callback_data,
   long int *&starter,
```

```
    int &starter_sz,
    groups::sims *&Stab,
    groups::strong_generators *&Strong_gens,
    long int *&candidates,
    int &nb_candidates,
    int &nb_cases,
    int verbose_level);
void read_orbit_rep_and_candidates_from_files(
       std::string &prefix,
    int level, int orbit_at_level, int level_of_candidates_file,
    long int *&starter,
    int &starter_sz,
    groups::sims *&Stab,
    groups::strong_generators *&Strong_gens,
    long int *&candidates,
    int &nb_candidates,
    int &nb_cases,
    int verbose_level);
void read_representatives(
       std::string &fname,
    int *&Reps, int &nb_reps, int &size, int verbose_level);
void read_representatives_and_strong_generators(
       std::string &fname,
    int *&Reps,
    char **&Aut_ascii, int &nb_reps,
    int &size, int verbose_level);
void read_file_and_print_representatives(
       std::string &fname,
    int f_print_stabilizer_generators, int verbose_level);
void read_set_and_stabilizer(
       std::string &fname,
    int no, long int *&set, int &set_sz, groups::sims *&stab,
    groups::strong_generators *&Strong_gens,
    int &nb_cases,
    int verbose_level);
void list_elements_as_permutations_vertically(
       data_structures_groups::vector_ge *gens,
       std::ostream &ost);
void print_symmetry_group_type(std::ostream &ost);
void print_info();
void report_basic_orbits(std::ostream &ost);
void print_base();
void print_bare_base(std::ofstream &ost);
void latex_all_points(std::ostream &ost);
void latex_point_set(
       std::ostream &ost,
       long int *set, int sz, int verbose_level);
```

```
void print_group_order(std::ostream &ost);
void print_group_order_long(std::ostream &ost);
void print_vector(
      data_structures_groups::vector_ge &v);
void print_vector_as_permutation(
      data_structures_groups::vector_ge &v);
void write_set_of_elements_latex_file(
      std::string &fname,
      std::string &title, int *Elt, int nb_elts);
void export_to_orbiter(
      std::string &fname, std::string &label,
      groups::strong_generators *SG, int verbose_level);
void export_to_orbiter_as_bsgs(
      std::string &fname,
      std::string &label,
      std::string &label_tex,
      groups::strong_generators *SG, int verbose_level);
void print_one_element_tex(
      std::ostream &ost,
      int *Elt, int f_with_permutation);

// action_cb.cpp
int image_of(void *elt, int a);
void image_of_low_level(void *elt,
      int *input, int *output, int verbose_level);
int linear_entry_ij(void *elt, int i, int j);
int linear_entry_frobenius(void *elt);
void one(void *elt);
int is_one(void *elt);
void unpack(void *elt, void *Elt);
void pack(void *Elt, void *elt);
void retrieve(void *elt, int hdl);
int store(void *elt);
void mult(void *a, void *b, void *ab);
void mult_apply_from_the_right(void *a, void *b);
   // a := a * b
void mult_apply_from_the_left(void *a, void *b);
   // b := a * b
void invert(void *a, void *av);
void invert_in_place(void *a);
void transpose(void *a, void *at);
void move(void *a, void *b);
void dispose(int hdl);
void print(
      std::ostream &ost, void *elt);
void print_quick(
      std::ostream &ost, void *elt);
```

```
void print_as_permutation(
      std::ostream &ost, void *elt);
void print_point(
      int a, std::ostream &ost);
void unrank_point(long int rk, int *v);
long int rank_point(int *v);
void code_for_make_element(
      int *data, void *elt);
void print_for_make_element(
      std::ostream &ost, void *elt);
void print_for_make_element_no_commas(
      std::ostream &ost, void *elt);

long int element_image_of(long int a, void *elt, int verbose_level);
void element_image_of_low_level(int *input, int *output,
   void *elt, int verbose_level);
int element_linear_entry_ij(void *elt, int i, int j,
   int verbose_level);
int element_linear_entry_frobenius(void *elt, int verbose_level);
void element_one(void *elt, int verbose_level);
int element_is_one(void *elt, int verbose_level);
void element_unpack(void *elt, void *Elt, int verbose_level);
void element_pack(void *Elt, void *elt, int verbose_level);
void element_retrieve(int hdl, void *elt, int verbose_level);
int element_store(void *elt, int verbose_level);
void element_mult(void *a, void *b, void *ab, int verbose_level);
void element_invert(void *a, void *av, int verbose_level);
void element_transpose(void *a, void *at, int verbose_level);
void element_move(void *a, void *b, int verbose_level);
void element_dispose(int hdl, int verbose_level);
void element_print(void *elt, std::ostream &ost);
void element_print_quick(void *elt, std::ostream &ost);
void element_print_latex(void *elt, std::ostream &ost);
void element_print_latex_with_extras(
      void *elt, std::string &label, std::ostream &ost);
void element_print_latex_with_print_point_function(
   void *elt, std::ostream &ost,
   void (*point_label)(std::stringstream &sstr,
         long int pt, void *data),
   void *point_label_data);
void element_print_verbose(void *elt, std::ostream &ost);
void element_code_for_make_element(void *elt, int *data);
void element_print_for_make_element(void *elt,
      std::ostream &ost);
void element_print_for_make_element_no_commas(void *elt,
      std::ostream &ost);
void element_print_as_permutation(void *elt,
```

```
      std::ostream &ost);
void element_as_permutation(void *elt,
   int *perm, int verbose_level);
void element_print_as_permutation_verbose(void *elt,
      std::ostream &ost, int verbose_level);
void element_print_as_permutation_with_offset(void *elt,
      std::ostream &ost,
   int offset, int f_do_it_anyway_even_for_big_degree,
   int f_print_cycles_of_length_one,
   int verbose_level);
void element_print_as_permutation_with_offset_and_max_cycle_length(
   void *elt,
   std::ostream &ost, int offset, int max_cycle_length,
   int f_orbit_structure);
void element_print_image_of_set(void *elt,
   int size, long int *set);
int element_signum_of_permutation(void *elt);
void element_write_file_fp(int *Elt,
      std::ofstream &fp, int verbose_level);
void element_read_file_fp(int *Elt,
      std::ifstream &fp, int verbose_level);
void element_write_file(int *Elt,
      std::string &fname, int verbose_level);
void element_read_file(int *Elt,
      std::string &fname, int verbose_level);
void element_write_to_memory_object(
      int *Elt,
      orbiter_kernel_system::memory_object *m,
      int verbose_level);
void element_read_from_memory_object(
      int *Elt,
      orbiter_kernel_system::memory_object *m,
      int verbose_level);
void element_write_to_file_binary(int *Elt,
      std::ofstream &fp, int verbose_level);
void element_read_from_file_binary(int *Elt,
      std::ifstream &fp, int verbose_level);
void random_element(groups::sims *S, int *Elt,
   int verbose_level);
void all_elements(
      data_structures_groups::vector_ge *&vec,
      int verbose_level);
void all_elements_save_csv(
      std::string &fname, int verbose_level);


// in backtrack.cpp
```

```
   int is_minimal(
      int size, long int *set, int &backtrack_level,
      int verbose_level);
   void make_canonical(
      int size, long int *set,
      long int *canonical_set, int *transporter,
      int &total_backtrack_nodes,
      int f_get_automorphism_group, groups::sims *Aut,
      int verbose_level);
   int is_minimal_witness(
      int size, long int *set,
      int &backtrack_level, long int *witness,
      int *transporter_witness,
      int &backtrack_nodes,
      int f_get_automorphism_group, groups::sims &Aut,
      int verbose_level);
};


// ##############################################################################
// action_global.cpp
// ##############################################################################

//! global functions related to group actions

class action_global {
public:
   void action_print_symmetry_group_type(
         std::ostream &ost, symmetry_group_type a);
   void get_symmetry_group_type_text(
         std::string &txt, std::string &tex,
         symmetry_group_type a);
   void make_generators_stabilizer_of_three_components(
      action *A_PGL_n_q, action *A_PGL_k_q,
      int k, data_structures_groups::vector_ge *gens,
      int verbose_level);
   void make_generators_stabilizer_of_two_components(
      action *A_PGL_n_q, action *A_PGL_k_q,
      int k, data_structures_groups::vector_ge *gens,
      int verbose_level);
   // used in semifield
   void compute_generators_GL_n_q(
         int *&Gens, int &nb_gens,
      int &elt_size, int n,
      field_theory::finite_field *F,
      data_structures_groups::vector_ge *&nice_gens,
      int verbose_level);
```

```
void set_orthogonal_group_type(
      int f_siegel,
   int f_reflection, int f_similarity,
   int f_semisimilarity);
int get_orthogonal_group_type_f_reflection();
void lift_generators(
      data_structures_groups::vector_ge *gens_in,
      data_structures_groups::vector_ge *&gens_out,
   action *Aq,
   field_theory::subfield_structure *S, int n,
   int verbose_level);
void retract_generators(
      data_structures_groups::vector_ge *gens_in,
      data_structures_groups::vector_ge *&gens_out,
   action *AQ,
   field_theory::subfield_structure *S, int n,
   int verbose_level);
void lift_generators_to_subfield_structure(
   int n, int s,
   field_theory::subfield_structure *S,
   action *Aq, action *AQ,
   groups::strong_generators *&Strong_gens,
   int verbose_level);
void perm_print_cycles_sorted_by_length(
      std::ostream &ost,
   int degree, int *perm, int verbose_level);
void perm_print_cycles_sorted_by_length_offset(
      std::ostream &ost,
   int degree, int *perm, int offset,
   int f_do_it_anyway_even_for_big_degree,
   int f_print_cycles_of_length_one, int verbose_level);
action *init_direct_product_group_and_restrict(
      groups::matrix_group *M1,
      groups::matrix_group *M2,
      int verbose_level);
action *init_direct_product_group(
      groups::matrix_group *M1,
      groups::matrix_group *M2,
      int verbose_level);
void compute_decomposition_based_on_orbits(
      geometry::projective_space *P,
      groups::schreier *Sch1, groups::schreier *Sch2,
      geometry::incidence_structure *&Inc,
      data_structures::partitionstack *&Stack,
      int verbose_level);
void compute_decomposition_based_on_orbit_length(
      geometry::projective_space *P,
```

```
      groups::schreier *Sch1, groups::schreier *Sch2,
      geometry::incidence_structure *&Inc,
      data_structures::partitionstack *&Stack,
      int verbose_level);
void orbits_on_equations(
      action *A,
      ring_theory::homogeneous_polynomial_domain *HPD,
   int *The_equations,
   int nb_equations, groups::strong_generators *gens,
   actions::action *&A_on_equations, groups::schreier *&Orb,
   int verbose_level);
void compute_fixed_objects_in_PG(
      int up_to_which_rank,
      action *A,
      geometry::projective_space *P,
   int *Elt,
   std::vector<std::vector<long int> > &Fix,
   int verbose_level);
void report_fixed_objects_in_P3(
      std::ostream &ost,
      action *A,
      geometry::projective_space *P3,
   int *Elt,
   int verbose_level);
groups::strong_generators *set_stabilizer_in_projective_space(
      action *A_linear,
      geometry::projective_space *P,
   long int *set, int set_size, //int &canonical_pt,
   int *canonical_set_or_NULL,
   int verbose_level);
// assuming we are in a linear action.
void stabilizer_of_dual_hyperoval_representative(
      action *A,
      int k, int n, int no,
      data_structures_groups::vector_ge *&gens,
      std::string &stab_order,
      int verbose_level);
void stabilizer_of_spread_representative(
      action *A,
      int q, int k, int no,
      data_structures_groups::vector_ge *&gens,
      std::string &stab_order,
      int verbose_level);
void stabilizer_of_quartic_curve_representative(
      action *A,
      int q, int no,
      data_structures_groups::vector_ge *&gens,
```

```
      std::string &stab_order,
      int verbose_level);
void perform_tests(
      action *A,
      groups::strong_generators *SG,
      int verbose_level);
void apply_based_on_text(
      action *A,
      std::string &input_text,
      std::string &input_group_element,
      int verbose_level);
void multiply_based_on_text(
      action *A,
      std::string &data_A,
      std::string &data_B, int verbose_level);
void inverse_based_on_text(
      action *A,
      std::string &data_A, int verbose_level);
void consecutive_powers_based_on_text(
      action *A,
      std::string &data_A,
      std::string &exponent_text, int verbose_level);
void raise_to_the_power_based_on_text(
      action *A,
      std::string &data_A,
      std::string &exponent_text, int verbose_level);
void compute_orbit_of_point(
      actions::action *A,
      data_structures_groups::vector_ge &strong_generators,
      int pt, int *orbit, int &len, int verbose_level);
void compute_orbit_of_point_generators_by_handle(
      actions::action *A,
      int nb_gen,
   int *gen_handle, int pt, int *orbit, int &len,
   int verbose_level);
int least_image_of_point(
      actions::action *A,
      data_structures_groups::vector_ge &strong_generators,
   int pt, int *transporter, int verbose_level);
int least_image_of_point_generators_by_handle(
      actions::action *A,
   std::vector<int> &gen_handle,
   int pt, int *transporter, int verbose_level);
int least_image_of_point_generators_by_handle(
      actions::action *A,
   int nb_gen, int *gen_handle,
   int pt, int *transporter, int verbose_level);
```

```
   void all_point_orbits(
         actions::action *A,
         groups::schreier &Schreier, int verbose_level);
   void all_point_orbits_from_generators(
         actions::action *A,
         groups::schreier &Schreier,
         groups::strong_generators *SG,
         int verbose_level);
   void all_point_orbits_from_single_generator(
         actions::action *A,
         groups::schreier &Schreier,
         int *Elt,
         int verbose_level);
   void compute_set_orbit(
         actions::action *A,
         data_structures_groups::vector_ge &gens,
      int size, long int *set,
      int &nb_sets, long int **&Sets, int **&Transporter,
      int verbose_level);
   void delete_set_orbit(
         actions::action *A,
         int nb_sets, long int **Sets, int **Transporter);
   void compute_minimal_set(
         actions::action *A,
         data_structures_groups::vector_ge &gens,
         int size, long int *set,
      long int *minimal_set, int *transporter,
      int verbose_level);

};

void callback_choose_random_generator_orthogonal(int iteration,
   int *Elt, void *data, int verbose_level);
   // for use in action_init.cpp


// ##############################################################################
// action_pointer_table.cpp
// ##############################################################################


//! interface to the implementation functions for group actions

class action_pointer_table {

public:
```

```
   std::string label;

   /** function pointers for group actions */
   long int (*ptr_element_image_of)(action &A, long int a, void *elt, int verbose_
level);
   void (*ptr_element_image_of_low_level)(action &A, int *input, int *output, void
 *elt, int verbose_level);
   int (*ptr_element_linear_entry_ij)(action &A, void *elt, int i, int j, int verb
ose_level);
   int (*ptr_element_linear_entry_frobenius)(action &A, void *elt, int verbose_lev
el);
   void (*ptr_element_one)(action &A, void *elt, int verbose_level);
   int (*ptr_element_is_one)(action &A, void *elt, int verbose_level);
   void (*ptr_element_unpack)(action &A, void *elt, void *Elt, int verbose_level);

   void (*ptr_element_pack)(action &A, void *Elt, void *elt, int verbose_level);
   void (*ptr_element_retrieve)(action &A, int hdl, void *elt, int verbose_level);

   int (*ptr_element_store)(action &A, void *elt, int verbose_level);
   void (*ptr_element_mult)(action &A, void *a, void *b, void *ab, int verbose_lev
el);
   void (*ptr_element_invert)(action &A, void *a, void *av, int verbose_level);
   void (*ptr_element_transpose)(action &A, void *a, void *at, int verbose_level);

   void (*ptr_element_move)(action &A, void *a, void *b, int verbose_level);
   void (*ptr_element_dispose)(action &A, int hdl, int verbose_level);
   void (*ptr_element_print)(action &A, void *elt, std::ostream &ost);
   void (*ptr_element_print_quick)(action &A, void *elt, std::ostream &ost);
   void (*ptr_element_print_latex)(action &A, void *elt, std::ostream &ost);
   void (*ptr_element_print_latex_with_print_point_function)(action &A,
      void *elt, std::ostream &ost,
      void (*point_label)(std::stringstream &sstr, long int pt, void *data),
      void *point_label_data);
   void (*ptr_element_print_verbose)(action &A, void *elt, std::ostream &ost);
   void (*ptr_print_point)(action &A, long int i, std::ostream &ost);
   void (*ptr_element_code_for_make_element)(action &A, void *elt, int *data);
   void (*ptr_element_print_for_make_element)(action &A, void *elt, std::ostream &
ost);
   void (*ptr_element_print_for_make_element_no_commas)(action &A,
      void *elt, std::ostream &ost);
   void (*ptr_unrank_point)(action &A, long int rk, int *v);
   long int (*ptr_rank_point)(action &A, int *v);

   /** counters for how often a function has been called */
   int nb_times_image_of_called;
   int nb_times_image_of_low_level_called;
   int nb_times_unpack_called;
```

```
   int nb_times_pack_called;
   int nb_times_retrieve_called;
   int nb_times_store_called;
   int nb_times_mult_called;
   int nb_times_invert_called;

   action_pointer_table();
   ~action_pointer_table();
   void null_function_pointers();
   void init_function_pointers_matrix_group();
   void init_function_pointers_wreath_product_group();
   void init_function_pointers_direct_product_group();
   void init_function_pointers_permutation_group();
   void init_function_pointers_permutation_representation_group();
   void init_function_pointers_induced_action();
};




// ##############################################################################
// known_groups.cpp
// ##############################################################################

//! creating a known group with a default action in an action object

class known_groups {
public:

   action *A;

   known_groups();
   ~known_groups();
   void init(action *A, int verbose_level);

   /** Create any linear group */
   void init_linear_group(
         field_theory::finite_field *F, int m,
      int f_projective, int f_general, int f_affine,
      int f_semilinear, int f_special,
      data_structures_groups::vector_ge *&nice_gens,
      int verbose_level);

   /** Create the projective linear (or semilinear) group PGL (or PGGL)*/
   void init_projective_group(
         int n, field_theory::finite_field *F,
```

```
   int f_semilinear, int f_basis, int f_init_sims,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);


/** Create the affine group AGL(n,q) */
void init_affine_group(
      int n, field_theory::finite_field *F,
   int f_semilinear,
   int f_basis, int f_init_sims,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);

/** Create the general linear group GL(n,q) */
void init_general_linear_group(
      int n, field_theory::finite_field *F,
   int f_semilinear, int f_basis, int f_init_sims,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);

void setup_linear_group_from_strong_generators(
      groups::matrix_group *M,
      data_structures_groups::vector_ge *&nice_gens,
      int f_init_sims,
   int verbose_level);

void init_sims_from_generators(int verbose_level);

/** Create the projective special linear group PSL */
void init_projective_special_group(
      int n, field_theory::finite_field *F,
   int f_semilinear, int f_basis, int verbose_level);

void init_matrix_group_strong_generators_builtin(
      groups::matrix_group *M,
      data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);
void init_permutation_group(
      int degree, int f_no_base, int verbose_level);
void init_permutation_group_from_nauty_output(
      data_structures::nauty_output *NO,
   int verbose_level);
void init_permutation_group_from_generators(
      int degree,
   int f_target_go, ring_theory::longinteger_object &target_go,
   int nb_gens, int *gens,
   int given_base_length, long int *given_base,
```

```
   int f_no_base,
   int verbose_level);

/** Create the affine group AGL(n,q) as abstract permutation group,
 * not as matrix group */
void init_affine_group(
      int n, int q, int f_translations,
   int f_semilinear, int frobenius_power,
   int f_multiplication,
   int multiplication_order, int verbose_level);


/** Create the symmetric group
 * as abstract permutation group */
void init_symmetric_group(
      int degree, int f_no_base, int verbose_level);
void init_cyclic_group(
      int degree, int f_no_base, int verbose_level);
void init_identity_group(
      int degree, int f_no_base, int verbose_level);


void create_sims(int verbose_level);
void create_orthogonal_group(
      action *subaction,
   int f_has_target_group_order,
   ring_theory::longinteger_object &target_go,
   void (* callback_choose_random_generator)(int iteration,
      int *Elt, void *data, int verbose_level),
   int verbose_level);

void init_orthogonal_group_with_O(
      orthogonal_geometry::orthogonal *O,
   int f_on_points, int f_on_lines, int f_on_points_and_lines,
   int f_semilinear,
   int f_basis, int verbose_level);


/** Create the wreath product group AGL(n,q) wreath Sym(nb_factors)
 * in wreath product action
 * and restrict the action to the tensor space. */
void init_wreath_product_group_and_restrict(
      int nb_factors, int n,
      field_theory::finite_field *F,
      data_structures_groups::vector_ge *&nice_gens,
      int verbose_level);
```

```
   /** Create the wreath product group AGL(n,q) wreath Sym(nb_factors)
    * in wreath product action
    */
   void init_wreath_product_group(
         int nb_factors, int n,
         field_theory::finite_field *F,
         data_structures_groups::vector_ge *&nice_gens,
         int verbose_level);


   /** Create the permutation representation with a given set of generators
    */
   void init_permutation_representation(
         action *A_original,
         int f_stay_in_the_old_action,
         data_structures_groups::vector_ge *gens,
         int *Perms, int degree,
         int verbose_level);

   /** Create the orthogonal group O(5,q) */
   void init_BLT(
         field_theory::finite_field *F, int f_basis,
      int f_init_hash_table, int verbose_level);


   /** Create a group from generators */
   void init_group_from_strong_generators(
         data_structures_groups::vector_ge *gens,
         groups::sims *K,
      int given_base_length, int *given_base,
      int verbose_level);


   /** Create the orthogonal group O^epsilon(n,q) */
   void init_orthogonal_group(
         int epsilon,
      int n, field_theory::finite_field *F,
      int f_on_points, int f_on_lines,
      int f_on_points_and_lines,
      int f_semilinear,
      int f_basis, int verbose_level);


};
```

```
// ##############################################################################
// stabilizer_chain_base_data.cpp:
// ##############################################################################



#define STABILIZER_CHAIN_DATA_MAX_DEGREE 1L << 29


//! base and transversals in a stabilizer chain for a permutation group


class stabilizer_chain_base_data {
private:
   action *A;

   /** whether we have a base (b_0,\ldots,b_{l-1}) */
   int f_has_base;

   /** the length of the base */
   int base_len;



   /** the base (b_0,\ldots,b_{l-1}) */
   long int *base;



   /** the length of the orbit of $G^{(i)}$ on $b_i$ */
   int *transversal_length;

   /** the orbit of  b_i */
   long int **orbit;

   /** the inverse orbit of  b_i */
   long int **orbit_inv;

   int *path;
public:

   stabilizer_chain_base_data();
   ~stabilizer_chain_base_data();
   void free_base_data();
   void allocate_base_data(
         action *A,
```

```
        int base_len, int verbose_level);
   void reallocate_base(int new_base_point);
   void init_base_from_sims(
        groups::sims *G, int verbose_level);
   int &get_f_has_base();
   int &get_base_len();
   long int &base_i(int i);
   long int *&get_base();
   int &transversal_length_i(int i);
   int *&get_transversal_length();
   long int &orbit_ij(int i, int j);
   long int &orbit_inv_ij(int i, int j);
   int &path_i(int i);
   void group_order(
        ring_theory::longinteger_object &go);
   void init_projective_matrix_group(
        field_theory::finite_field *F,
        int n, int f_semilinear, int degree,
        int verbose_level);
   void init_affine_matrix_group(
        field_theory::finite_field *F,
        int n, int f_semilinear, int degree,
        int verbose_level);
   void init_linear_matrix_group(
        field_theory::finite_field *F,
        int n, int f_semilinear, int degree,
        int verbose_level);
};



}}}



#endif /* ORBITER_SRC_LIB_GROUP_ACTIONS_ACTIONS_ACTIONS_H_ */
```

5.3 Data Structures

```
// data_structures.h
//
// Anton Betten
//
// moved here from action.h: July 28, 2018
// based on action.h which was started:  August 13, 2005


#ifndef ORBITER_SRC_LIB_GROUP_ACTIONS_DATA_STRUCTURES_DATA_STRUCTURES_H_
#define ORBITER_SRC_LIB_GROUP_ACTIONS_DATA_STRUCTURES_DATA_STRUCTURES_H_



namespace orbiter {
namespace layer3_group_actions {
namespace data_structures_groups {




// ##############################################################################
// group_container.cpp
// ##############################################################################


//! a container data structure for groups




class group_container {

public:
   actions::action *A;

   int f_has_ascii_coding;
   std::string ascii_coding;

   int f_has_strong_generators;
   vector_ge *SG;
   int *tl;

   int f_has_sims;
   groups::sims *S;

   group_container();
```

```
~group_container();
void init(actions::action *A,
      int verbose_level);
void init_ascii_coding_to_sims(
      std::string &ascii_coding, int verbose_level);
void init_ascii_coding(
      std::string &ascii_coding, int verbose_level);
void delete_ascii_coding();
void delete_sims();
void init_strong_generators_empty_set(int verbose_level);
void init_strong_generators(vector_ge &SG,
      int *tl, int verbose_level);
void init_strong_generators_by_handle_and_with_tl(
      std::vector<int> &gen_handle,
      std::vector<int> &tl, int verbose_level);
void init_strong_generators_by_hdl(
      int nb_gen, int *gen_hdl,
   int *tl, int verbose_level);
void delete_strong_generators();
void require_ascii_coding();
void require_strong_generators();
void require_sims();
void group_order(ring_theory::longinteger_object &go);
void print_group_order(std::ostream &ost);
void print_tl();
void code_ascii(int verbose_level);
void decode_ascii(int verbose_level);
void schreier_sims(int verbose_level);
void get_strong_generators(int verbose_level);
void point_stabilizer(
      group_container &stab, int pt,
      int verbose_level);
void point_stabilizer_with_action(actions::action *A2,
      group_container &stab, int pt, int verbose_level);
void induced_action(
      actions::action &induced_action,
      group_container &H, group_container &K,
      int verbose_level);
void extension(group_container &N,
      group_container &H, int verbose_level);
   // N needs to have strong generators,
   // H needs to have sims
   // N and H may have different actions,
   // the action of N is taken for the extension.
void print_strong_generators(std::ostream &ost,
   int f_print_as_permutation);
void print_strong_generators_with_different_action(
```

```
        std::ostream &ost, actions::action *A2);
   void print_strong_generators_with_different_action_verbose(
        std::ostream &ost,
        actions::action *A2,
        int verbose_level);

};


// ##############################################################################
// incidence_structure_with_group.cpp
// ##############################################################################



//! to represent an incidence structure and its group


class incidence_structure_with_group {

public:

   geometry::incidence_structure *Inc;
   int N; // Inc->nb_rows + Inc->nb_cols;

   int *partition;

   int f_has_canonical_form;
   data_structures::bitvector *canonical_form;

   int f_has_canonical_labeling;
   long int *canonical_labeling;  // [nb_rows + nb_cols]

   actions::action *A_perm; // degree = N

   incidence_structure_with_group();
   ~incidence_structure_with_group();
   void init(geometry::incidence_structure *Inc,
      int *partition,
      int verbose_level);
   void set_stabilizer_and_canonical_form(
        int f_compute_canonical_form,
        geometry::incidence_structure *&Inc_out,
        int verbose_level);
};
```

```
// ##############################################################################
// orbit_rep.cpp
// ##############################################################################


//! to hold one orbit after reading files from Orbiters poset classification


class orbit_rep {
public:
   actions::action *A;
   void (*early_test_func_callback)(long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      void *data, int verbose_level);
   void *early_test_func_callback_data;

   int level;
   int orbit_at_level;
   int nb_cases;

   long int *rep;

   groups::sims *Stab;
   groups::strong_generators *Strong_gens;

   ring_theory::longinteger_object *stab_go;
   long int *candidates;
   int nb_candidates;


   orbit_rep();
   ~orbit_rep();
   void init_from_file(actions::action *A,
         std::string &prefix,
      int level, int orbit_at_level,
      int level_of_candidates_file,
      void (*early_test_func_callback)(long int *S, int len,
         long int *candidates, int nb_candidates,
         long int *good_candidates, int &nb_good_candidates,
         void *data, int verbose_level),
      void *early_test_func_callback_data,
      int verbose_level);

};
```

```
// ##############################################################################
// orbit_transversal.cpp
// ##############################################################################

//! a set of orbits using a vector of orbit representatives and stabilizers


class orbit_transversal {

public:
   actions::action *A;
   actions::action *A2;

   int nb_orbits;
   set_and_stabilizer *Reps;

   orbit_transversal();
   ~orbit_transversal();
   void init_from_schreier(
         groups::schreier *Sch,
         actions::action *default_action,
         ring_theory::longinteger_object &full_group_order,
         int verbose_level);
   void read_from_file(actions::action *A,
         actions::action *A2,
         std::string &fname, int verbose_level);
   void read_from_file_one_case_only(
         actions::action *A,
         actions::action *A2,
         std::string &fname,
         int case_nr,
         set_and_stabilizer *&Rep,
         int verbose_level);
   data_structures::tally *get_ago_distribution(long int *&ago,
         int verbose_level);
   void report_ago_distribution(std::ostream &ost);
   void print_table_latex(
         std::ostream &f,
         int f_has_callback,
         void (*callback_print_function)(
               std::stringstream &ost, void *data, void *callback_data),
         void *callback_data,
         int f_has_callback2,
         void (*callback_print_function2)(
               std::stringstream &ost, void *data, void *callback_data),
```

```
         void *callback_data2,
         int verbose_level);
   void export_data_in_source_code_inside_tex(
         std::string &prefix,
         std::string &label_of_structure, std::ostream &ost,
         int verbose_level);
};



// ##############################################################################
// orbit_type_repository.cpp
// ##############################################################################





//! A collection of invariants called orbit type associated with a system of sets
. The orbit types are based on the orbits of a given group.



class orbit_type_repository {

public:

   groups::orbits_on_something *Oos;

   int nb_sets;
   int set_size;
   long int *Sets; // [nb_sets * set_size]
      // A system of sets that is given
   long int goi;

   int orbit_type_size;
      // the size of the invariant
   long int *Type_repository; // [nb_sets * orbit_type_size]
      // for each set, the orbit invariant

      // The next items are related to the classification of the
      // orbit invariant:

   int nb_types;
      // the number of distinct types that appear in the Type_repository
   int *type_first; // [nb_types]
   int *type_len; // [nb_types]
```

```
   int *type; // [nb_sets]
      // type[i] is the index into the Type_representatives of the
      // invariant associated with the i-th set in Sets[]
   long int *Type_representatives; // [nb_types]
      // The distinct types that appear in the Type_repository

   orbit_type_repository();
   ~orbit_type_repository();
   void init(
         groups::orbits_on_something *Oos,
         int nb_sets,
         int set_size,
         long int *Sets,
         long int goi,
         int verbose_level);
   void create_latex_report(
         std::string &prefix, int verbose_level);
   void report(std::ostream &ost, int verbose_level);
   void report_one_type(std::ostream &ost,
         int type_idx, int verbose_level);

};





// ##############################################################################
// schreier_vector_handler.cpp:
// ##############################################################################

//! manages access to schreier vectors


class schreier_vector_handler {
public:
   actions::action *A;
   actions::action *A2;
   int *cosetrep;
   int *Elt1;
   int *Elt2;
   int *Elt3;
   int f_check_image;
   int f_allow_failure;
   int nb_calls_to_coset_rep_inv;
   int nb_calls_to_coset_rep_inv_recursion;
```

```
   schreier_vector_handler();
   ~schreier_vector_handler();
   void init(actions::action *A, actions::action *A2,
         int f_allow_failure,
         int verbose_level);
   void print_info_and_generators(
         schreier_vector *S);
   int coset_rep_inv_lint(
         schreier_vector *S,
         long int pt, long int &pt0,
         int verbose_level);
   int coset_rep_inv(
         schreier_vector *S,
         int pt, int &pt0,
         int verbose_level);
   int coset_rep_inv_recursion(
      schreier_vector *S,
      int pt, int &pt0,
      int verbose_level);
   schreier_vector *sv_read_file(
         int gen_hdl_first, int nb_gen,
         std::ifstream &fp, int verbose_level);
   void sv_write_file(schreier_vector *Sv,
         std::ofstream &fp, int verbose_level);
   data_structures::set_of_sets *get_orbits_as_set_of_sets(
         schreier_vector *Sv,
         int verbose_level);

};

// ##############################################################################
// schreier_vector.cpp:
// ##############################################################################

//! compact storage of schreier vectors


class schreier_vector {
public:
   int gen_hdl_first;
   int nb_gen;
   int number_of_orbits;
   int *sv;
      // the length of sv is n+1 if the group is trivial
      // and 3*n + 1 otherwise.
      //
```

```
   // sv[0] = n = number of points in the set on which we act
   // the next n entries are the points of the set
   // the next 2*n entries only exist if the group is non-trivial:
   // the next n entries are the previous pointers
   // the next n entries are the labels
int f_has_local_generators;
vector_ge *local_gens;

schreier_vector();
~schreier_vector();
void init(int gen_hdl_first, int nb_gen, int *sv,
      int verbose_level);
void init_local_generators(
      vector_ge *gens,
      int verbose_level);
void set_sv(int *sv, int verbose_level);
int *points();
int *prev();
int *label();
int get_number_of_points();
int get_number_of_orbits();
int count_number_of_orbits();
void count_number_of_orbits_and_get_orbit_reps(
      int *&orbit_reps, int &nb_orbits);
int determine_depth_recursion(
   int n, int *pts, int *prev,
   int *depth, int *ancestor, int pos);
void relabel_points(
      induced_actions::action_on_factor_space *AF,
   int verbose_level);
void orbit_stats(
      int &nb_orbits, int *&orbit_reps,
      int *&orbit_length, int *&total_depth,
      int verbose_level);
void orbit_of_point(
      int pt, long int *&orbit_elts,
      int &orbit_len, int &idx_of_root_node,
      int verbose_level);
void init_from_schreier(groups::schreier *S,
   int f_trivial_group, int verbose_level);
void init_shallow_schreier_forest(groups::schreier *S,
   int f_trivial_group, int f_randomized,
   int verbose_level);
// initializes local_gens
void export_tree_as_layered_graph(
      int orbit_no, int orbit_rep,
      std::string &fname_mask,
```

```
        int verbose_level);
   void trace_back(int pt, int &depth);
   void print();
};



// ##############################################################################
// set_and_stabilizer.cpp
// ##############################################################################


//! a set and its known set stabilizer



class set_and_stabilizer {

public:
   actions::action *A;
   actions::action *A2;
   long int *data;
   int sz;
   ring_theory::longinteger_object target_go;
   groups::strong_generators *Strong_gens;
   groups::sims *Stab;

   set_and_stabilizer();
   ~set_and_stabilizer();
   void init(actions::action *A,
         actions::action *A2,
         int verbose_level);
   void group_order(ring_theory::longinteger_object &go);
   long int group_order_as_lint();
   void init_everything(actions::action *A,
         actions::action *A2,
         long int *Set, int set_sz,
         groups::strong_generators *gens,
         int verbose_level);
   void allocate_data(int sz, int verbose_level);
   set_and_stabilizer *create_copy(int verbose_level);
   void init_data(long int *data, int sz, int verbose_level);
   void init_stab_from_data(int *data_gens,
      int data_gens_size, int nb_gens,
      std::string &ascii_target_go,
      int verbose_level);
   void init_stab_from_file(const char *fname_gens,
```

```
      int verbose_level);
   void print_set_tex(std::ostream &ost);
   void print_set_tex_for_inline_text(std::ostream &ost);
   void print_generators_tex(std::ostream &ost);
   void apply_to_self(
         int *Elt, int verbose_level);
   void apply_to_self_inverse(
         int *Elt, int verbose_level);
   void apply_to_self_element_raw(
         int *Elt_data, int verbose_level);
   void apply_to_self_inverse_element_raw(int *Elt_data,
      int verbose_level);
   void rearrange_by_orbits(int *&orbit_first,
      int *&orbit_length, int *&orbit,
      int &nb_orbits,
      int verbose_level);
   actions::action *create_restricted_action_on_the_set(
         int verbose_level);
   void print_restricted_action_on_the_set(int verbose_level);
   void test_if_group_acts(int verbose_level);
   int find(long int pt);
};


// ##############################################################################
// translation_plane_via_andre_model.cpp
// ##############################################################################

//! Andre / Bruck / Bose model of a translation plane



class translation_plane_via_andre_model {
public:

   std::string label_txt;
   std::string label_tex;

   field_theory::finite_field *F;
   int q;
   int k;
   int n;
   int k1;
   int n1;
   int order_of_plane;

   geometry::andre_construction *Andre;
```

```
   int N; // number of points = number of lines
   int twoN; // 2 * N
   int f_semilinear;

   geometry::andre_construction_line_element *Line;
   int *Incma;
   int *pts_on_line;
   int *Line_through_two_points; // [N * N]
   int *Line_intersection; // [N * N]

   actions::action *An;
   actions::action *An1;

   actions::action *OnAndre;

   groups::strong_generators *strong_gens;

   geometry::incidence_structure *Inc;
   data_structures::partitionstack *Stack;

   translation_plane_via_andre_model();
   ~translation_plane_via_andre_model();
   void init(
         int k,
         std::string &label_txt,
         std::string &label_tex,
         groups::strong_generators *Sg,
         geometry::andre_construction *Andre,
         actions::action *An,
         actions::action *An1,
         int verbose_level);
#if 0
   void classify_arcs(
         poset_classification::poset_classification_control *Control,
         int verbose_level);
   void classify_subplanes(
         poset_classification::poset_classification_control *Control,
         int verbose_level);
#endif
   int check_arc(long int *S, int len, int verbose_level);
   int check_subplane(long int *S, int len, int verbose_level);
   int check_if_quadrangle_defines_a_subplane(
      long int *S, int *subplane7,
      int verbose_level);
   void create_latex_report(int verbose_level);
   void report(std::ostream &ost, int verbose_level);
   void export_incma(int verbose_level);
```

```
   void p_rank(int p, int verbose_level);

};



// ##############################################################################
// union_find.cpp
// ##############################################################################


//! a union find data structure (used in the poset classification algorithm)




class union_find {

public:
   actions::action *A;
   int *prev;


   union_find();
   ~union_find();
   void init(actions::action *A, int verbose_level);
   int ancestor(int i);
   int count_ancestors();
   int count_ancestors_above(int i0);
   void do_union(int a, int b);
   void print();
   void add_generators(vector_ge *gens, int verbose_level);
   void add_generator(int *Elt, int verbose_level);
};

// ##############################################################################
// union_find_on_k_subsets.cpp
// ##############################################################################

//! a union find data structure (used in the poset classification algorithm)



class union_find_on_k_subsets {

public:
```

```
   long int *set;
   int set_sz;
   int k;

   groups::sims *S;

   long int *interesting_k_subsets;
   int nb_interesting_k_subsets;

   actions::action *A_original;
   actions::action *Ar; // restricted action on the set
   actions::action *Ar_perm;
   actions::action *Ark; // Ar_perm on k_subsets
   actions::action *Arkr; // Ark restricted to interesting_k_subsets

   vector_ge *gens_perm;

   union_find *UF;


   union_find_on_k_subsets();
   ~union_find_on_k_subsets();
   void init(actions::action *A_original, groups::sims *S,
      long int *set, int set_sz, int k,
      long int *interesting_k_subsets,
      int nb_interesting_k_subsets,
      int verbose_level);
   int is_minimal(int rk, int verbose_level);
};



// ##############################################################################
// vector_ge_description.cpp
// ##############################################################################



//! to define a vector of group elements


class vector_ge_description {

public:

   int f_action;
   std::string action_label;
```

```
   int f_read_csv;
   std::string read_csv_fname;
   std::string read_csv_column_label;

   vector_ge_description();
   ~vector_ge_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();
};




// ##############################################################################
// vector_ge.cpp
// ##############################################################################

//! to hold a vector of group elements


class vector_ge {

public:
   actions::action *A;
   int *data;
   int len;

   vector_ge();
   ~vector_ge();
   void init(actions::action *A, int verbose_level);
   void copy(vector_ge *&vector_copy, int verbose_level);
   void init_by_hdl(actions::action *A,
         int *gen_hdl, int nb_gen, int verbose_level);
   void init_single(actions::action *A,
         int *Elt, int verbose_level);
   void init_double(actions::action *A,
         int *Elt1, int *Elt2, int verbose_level);
   void init_from_permutation_representation(
         actions::action *A, groups::sims *S, int *data,
      int nb_elements, int verbose_level);
      // data[nb_elements * degree]
   void init_from_data(actions::action *A, int *data,
      int nb_elements, int elt_size, int verbose_level);
   void init_conjugate_svas_of(vector_ge *v, int *Elt,
```

```
   int verbose_level);
void init_conjugate_sasv_of(vector_ge *v, int *Elt,
   int verbose_level);
int *ith(int i);
void print(std::ostream &ost);
void print_quick(std::ostream& ost);
void print_tex(std::ostream &ost);
void print_generators_tex(
      ring_theory::longinteger_object &go,
      std::ostream &ost);
void print_as_permutation(std::ostream& ost);
void allocate(int length, int verbose_level);
void reallocate(int new_length, int verbose_level);
void reallocate_and_insert_at(
      int position, int *elt, int verbose_level);
void insert_at(int length_before,
      int position, int *elt, int verbose_level);
void append(int *elt, int verbose_level);
void copy_in(int i, int *elt);
void copy_out(int i, int *elt);
void conjugate_svas(int *Elt);
void conjugate_sasv(int *Elt);
void print_with_given_action(
      std::ostream &ost, actions::action *A2);
void print(std::ostream &ost,
      int f_print_as_permutation,
   int f_offset, int offset,
   int f_do_it_anyway_even_for_big_degree,
   int f_print_cycles_of_length_one);
void print_for_make_element(std::ostream &ost);
void write_to_memory_object(
      orbiter_kernel_system::memory_object *m,
   int verbose_level);
void read_from_memory_object(
      orbiter_kernel_system::memory_object *m,
   int verbose_level);
void write_to_file_binary(std::ofstream &fp,
   int verbose_level);
void read_from_file_binary(std::ifstream &fp,
   int verbose_level);
void write_to_csv_file_coded(
      std::string &fname, int verbose_level);
void save_csv(
      std::string &fname, int verbose_level);
void export_inversion_graphs(
      std::string &fname, int verbose_level);
void read_column_csv(std::string &fname,
```

```
        actions::action *A, int col_idx,
        int verbose_level);
   void read_column_csv_using_column_label(
        std::string &fname,
        actions::action *A,
        std::string &column_label,
        int verbose_level);
   void extract_subset_of_elements_by_rank_text_vector(
        std::string &rank_vector_text, groups::sims *S,
      int verbose_level);
   void extract_subset_of_elements_by_rank(
        int *rank_vector,
      int len, groups::sims *S, int verbose_level);
   int test_if_all_elements_stabilize_a_point(
        actions::action *A2, int pt);
   int test_if_all_elements_stabilize_a_set(
        actions::action *A2,
      long int *set, int sz,
      int verbose_level);
   groups::schreier *orbits_on_points_schreier(
        actions::action *A_given, int verbose_level);
   void reverse_isomorphism_exterior_square(int verbose_level);
   void matrix_representation(
        induced_actions::action_on_homogeneous_polynomials *A_on_HPD,
        int *&M, int &nb_gens,
        int verbose_level);

};



}}}


#endif /* ORBITER_SRC_LIB_GROUP_ACTIONS_DATA_STRUCTURES_DATA_STRUCTURES_H_ */
```

5.4 Groups

```
// group_theory.h
//
// Anton Betten
//
// moved here from action.h: July 28, 2018
// based on action.h which was started:  August 13, 2005



#ifndef ORBITER_SRC_LIB_GROUP_ACTIONS_GROUPS_GROUPS_H_
#define ORBITER_SRC_LIB_GROUP_ACTIONS_GROUPS_GROUPS_H_


namespace orbiter {
namespace layer3_group_actions {
namespace groups {



// ################################################################################
// direct_product.cpp
// ################################################################################

//! the direct product of two matrix groups in product action

class direct_product {

public:
   matrix_group *M1;
   matrix_group *M2;
   field_theory::finite_field *F1;
   field_theory::finite_field *F2;
   int q1;
   int q2;

   std::string label;
   std::string label_tex;


   int degree_of_matrix_group1;
   int dimension_of_matrix_group1;
   int degree_of_matrix_group2;
   int dimension_of_matrix_group2;
   int degree_of_product_action;
   int degree_overall;
   int low_level_point_size;
```

```
int make_element_size;
int elt_size_int;

int *perm_offset_i;
int *tmp_Elt1;

int bits_per_digit1;
int bits_per_digit2;

int bits_per_elt;
int char_per_elt;

uchar *elt1;

int base_len_in_component1;
long int *base_for_component1;
int *tl_for_component1;

int base_len_in_component2;
long int *base_for_component2;
int *tl_for_component2;

int base_length;
long int *the_base;
int *the_transversal_length;

data_structures::page_storage *Elts;

direct_product();
~direct_product();
void init(matrix_group *M1, matrix_group *M2,
      int verbose_level);
long int element_image_of(
      int *Elt, long int a, int verbose_level);
void element_one(int *Elt);
int element_is_one(int *Elt);
void element_mult(int *A, int *B, int *AB, int verbose_level);
void element_move(int *A, int *B, int verbose_level);
void element_invert(int *A, int *Av, int verbose_level);
int offset_i(int i);
void element_pack(int *Elt, uchar *elt);
void element_unpack(uchar *elt, int *Elt);
void put_digit(uchar *elt, int f, int i, int d);
int get_digit(uchar *elt, int f, int i);
void make_element(int *Elt, int *data, int verbose_level);
void element_print_easy(int *Elt, std::ostream &ost);
void compute_base_and_transversals(int verbose_level);
```

```
   void make_strong_generators_data(int *&data,
         int &size, int &nb_gens, int verbose_level);
   void lift_generators(
         strong_generators *SG1,
         strong_generators *SG2,
         actions::action *A, strong_generators *&SG3,
         int verbose_level);
};


// ##############################################################################
// exceptional_isomorphism_O4.cpp
// ##############################################################################

//! exceptional isomorphism between orthogonal groups: O4, O5 and GL(2,q)

class exceptional_isomorphism_O4 {
public:
   field_theory::finite_field *Fq;
   actions::action *A2;
   actions::action *A4;
   actions::action *A5;

   int *E5a;
   int *E4a;
   int *E2a;
   int *E2b;

   exceptional_isomorphism_O4();
   ~exceptional_isomorphism_O4();
   void init(field_theory::finite_field *Fq,
         actions::action *A2,
         actions::action *A4,
         actions::action *A5,
         int verbose_level);
   void apply_2to4_embedded(
      int f_switch, int *mtx2x2_T, int *mtx2x2_S, int *Elt,
      int verbose_level);
   void apply_5_to_4(
      int *mtx4x4, int *mtx5x5, int verbose_level);
   void apply_4_to_5(
      int *E4, int *E5, int verbose_level);
   void apply_4_to_2(
      int *E4, int &f_switch, int *E2_a, int *E2_b,
      int verbose_level);
   void apply_2_to_4(
      int &f_switch, int *E2_a, int *E2_b, int *E4,
```

```
      int verbose_level);
   void print_as_2x2(int *mtx4x4);
};




// ##############################################################################
// linear_group_description.cpp
// ##############################################################################

//! description of a linear group from the command line


class linear_group_description {
public:
   int f_projective;
   int f_general;
   int f_affine;
   int f_GL_d_q_wr_Sym_n;
   int f_orthogonal;
   int f_orthogonal_p;
   int f_orthogonal_m;
   int GL_wreath_Sym_d;
   int GL_wreath_Sym_n;


   int f_n;
   int n;

   std::string input_q;

   field_theory::finite_field *F;


   int f_semilinear;
   int f_special;

   // induced actions and subgroups:


   int f_wedge_action;
   int f_wedge_action_detached;
   int f_PGL2OnConic;
   int f_monomial_group;
   int f_diagonal_group;
   int f_null_polarity_group;
```

```
   int f_symplectic_group;
   int f_singer_group;
   int f_singer_group_and_frobenius;
   int singer_power;
   int f_subfield_structure_action;
   int s;
   int f_subgroup_from_file;
   int f_borel_subgroup_upper;
   int f_borel_subgroup_lower;
   int f_identity_group;
   std::string subgroup_fname;
   std::string subgroup_label;
   int f_orthogonal_group;
   int orthogonal_group_epsilon;

   int f_on_tensors;
   int f_on_rank_one_tensors;

   int f_subgroup_by_generators;
   std::string subgroup_order_text;
   int nb_subgroup_generators;
   std::string subgroup_generators_label;

   int f_Janko1;

   int f_export_magma;


   int f_import_group_of_plane;
   std::string import_group_of_plane_label;


   linear_group_description();
   ~linear_group_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// linear_group.cpp
// ##############################################################################

//! creates a linear group from command line arguments using linear_group_descrip
```

```
tion

class linear_group {
public:
   linear_group_description *description;
   int n;
   int input_q;
   field_theory::finite_field *F;
   int f_semilinear;

   std::string label;
   std::string label_tex;

   strong_generators *initial_strong_gens;
   actions::action *A_linear;
   matrix_group *Mtx;

   int f_has_strong_generators;
   strong_generators *Strong_gens;
   actions::action *A2;
   int vector_space_dimension;
   int q;

   int f_has_nice_gens;
   data_structures_groups::vector_ge *nice_gens;

   linear_group();
   ~linear_group();
   void linear_group_init(
         linear_group_description *description,
      int verbose_level);
   void linear_group_import(int verbose_level);
   void linear_group_import_group_of_plane(int verbose_level);
   void linear_group_create(int verbose_level);
   int linear_group_apply_modification(
         linear_group_description *description,
         int verbose_level);

   void init_PGL2q_OnConic(int verbose_level);
   void init_wedge_action(int verbose_level);
   void init_wedge_action_detached(int verbose_level);
   void init_monomial_group(int verbose_level);
   void init_diagonal_group(int verbose_level);
   void init_singer_group(int singer_power, int verbose_level);
   void init_singer_group_and_frobenius(
         int singer_power, int verbose_level);
   void init_null_polarity_group(int verbose_level);
```

```
   void init_borel_subgroup_upper(int verbose_level);
   void init_identity_subgroup(int verbose_level);
   void init_symplectic_group(int verbose_level);
   void init_subfield_structure_action(int s, int verbose_level);
   void init_orthogonal_group(int epsilon, int verbose_level);
   void init_subgroup_from_file(
         std::string &subgroup_fname,
         std::string &subgroup_label,
         int verbose_level);
   void init_subgroup_by_generators(
         std::string &subgroup_label,
         std::string &subgroup_order_text,
         int nb_subgroup_generators,
         int *subgroup_generators_data,
         int verbose_level);
   void init_subgroup_Janko1(int verbose_level);
   void report(
         std::ostream &fp,
         int f_sylow, int f_group_table,
         int f_conjugacy_classes_and_normalizers,
         graphics::layered_graph_draw_options *LG_Draw_options,
         int verbose_level);
   void create_latex_report(
         graphics::layered_graph_draw_options *O,
         int f_sylow, int f_group_table, int f_classes,
         int verbose_level);

};




// ##############################################################################
// matrix_group.cpp
// ##############################################################################

//! a matrix group over a finite field in projective, vector space or affine acti
on

class matrix_group {

public:
   int f_projective;
      // n x n matrices (possibly with Frobenius)
      // acting on PG(n - 1, q)
   int f_affine;
```

```
   // n x n matrices plus translations
   // (possibly with Frobenius)
   // acting on F_q^n
int f_general_linear;
   // n x n matrices (possibly with Frobenius)
   // acting on F_q^n

int n;
   // the size of the matrices

int degree;
   // the degree of the action:
   // (q^(n-1)-1) / (q - 1) if f_projective
   // q^n if f_affine or f_general_linear

int f_semilinear;
   // use Frobenius automorphism

int f_kernel_is_diagonal_matrices;

int bits_per_digit;
int bits_per_elt;
int bits_extension_degree;
int char_per_elt;
int elt_size_int;
int elt_size_int_half;
int low_level_point_size; // added Jan 26, 2010
   // = n, the size of the vectors on which we act
int make_element_size;


std::string label;
std::string label_tex;

int f_GFq_is_allocated;
   // if TRUE, GFq will be destroyed in the destructor
   // if FALSE, it is the responsibility
   // of someone else to destroy GFq

field_theory::finite_field *GFq;
void *data;

algebra::gl_classes *C; // added Dec 2, 2013


// temporary variables, do not use!
int *Elt1, *Elt2, *Elt3;
```

```
   // used for mult, invert
int *Elt4;
   // used for invert
int *Elt5;
int *tmp_M;
   // used for GL_mult_internal
int *base_cols;
   // used for Gauss during invert
int *v1, *v2;
   // temporary vectors of length 2n
int *v3;
   // used in GL_mult_vector_from_the_left_contragredient
uchar *elt1, *elt2, *elt3;
   // temporary storage, used in element_store()

data_structures::page_storage *Elts;


matrix_group();
~matrix_group();

void init_projective_group_label(int n,
      field_theory::finite_field *F,
      int f_semilinear, int f_special,
      actions::action *A,
      std::string &label,
      std::string &label_tex,
      int verbose_level);
void init_projective_group(int n,
      field_theory::finite_field *F,
   int f_semilinear, actions::action *A, int verbose_level);
void init_affine_group(int n,
      field_theory::finite_field *F,
   int f_semilinear, actions::action *A, int verbose_level);
void init_general_linear_group(int n,
      field_theory::finite_field *F,
   int f_semilinear, actions::action *A, int verbose_level);
void allocate_data(int verbose_level);
void free_data(int verbose_level);
void setup_page_storage(int page_length_log, int verbose_level);
void compute_elt_size(int verbose_level);
void init_base(actions::action *A, int verbose_level);
void init_base_projective(actions::action *A, int verbose_level);
   // initializes base, base_len, degree,
   // transversal_length, orbit, orbit_inv
void init_base_affine(actions::action *A, int verbose_level);
void init_base_general_linear(actions::action *A, int verbose_level);
```

```
void init_gl_classes(int verbose_level);

int GL_element_entry_ij(int *Elt, int i, int j);
int GL_element_entry_frobenius(int *Elt);
long int image_of_element(int *Elt, long int a, int verbose_level);
long int GL_image_of_PG_element(int *Elt, long int a, int verbose_level);
long int GL_image_of_AG_element(int *Elt, long int a, int verbose_level);
void action_from_the_right_all_types(
   int *v, int *A, int *vA, int verbose_level);
void projective_action_from_the_right(
   int *v, int *A, int *vA, int verbose_level);
void general_linear_action_from_the_right(
   int *v, int *A, int *vA, int verbose_level);
void substitute_surface_equation(int *Elt,
      int *coeff_in, int *coeff_out,
      algebraic_geometry::surface_domain *Surf,
      int verbose_level);
void GL_one(int *Elt);
void GL_one_internal(int *Elt);
void GL_zero(int *Elt);
int GL_is_one(int *Elt);
void GL_mult(int *A, int *B, int *AB, int verbose_level);
void GL_mult_internal(int *A, int *B, int *AB, int verbose_level);
void GL_copy(int *A, int *B);
void GL_copy_internal(int *A, int *B);
void GL_transpose(int *A, int *At, int verbose_level);
void GL_transpose_internal(int *A, int *At, int verbose_level);
void GL_invert(int *A, int *Ainv);
void GL_invert_internal(int *A, int *Ainv, int verbose_level);
void GL_unpack(uchar *elt, int *Elt, int verbose_level);
void GL_pack(int *Elt, uchar *elt, int verbose_level);
void GL_print_easy(int *Elt, std::ostream &ost);
void GL_code_for_make_element(int *Elt, int *data);
void GL_print_for_make_element(int *Elt, std::ostream &ost);
void GL_print_for_make_element_no_commas(
      int *Elt, std::ostream &ost);
void GL_print_easy_normalized(int *Elt, std::ostream &ost);
void GL_print_latex(int *Elt, std::ostream &ost);
void GL_print_latex_with_print_point_function(int *Elt,
      std::ostream &ost,
      void (*point_label)(
            std::stringstream &sstr, int pt, void *data),
      void *point_label_data);
void GL_print_easy_latex(int *Elt, std::ostream &ost);
void GL_print_easy_latex_with_option_numerical(
      int *Elt, int f_numerical, std::ostream &ost);
void decode_matrix(int *Elt, int n, uchar *elt);
```

```
   int get_digit(uchar *elt, int i, int j);
   int decode_frobenius(uchar *elt);
   void encode_matrix(int *Elt, int n, uchar *elt, int verbose_level);
   void put_digit(uchar *elt, int i, int j, int d);
   void encode_frobenius(uchar *elt, int d);
   void make_element(int *Elt, int *data, int verbose_level);
   void make_GL_element(int *Elt, int *A, int f);
   void orthogonal_group_random_generator(
         actions::action *A,
         orthogonal_geometry::orthogonal *O,
      int f_siegel,
      int f_reflection,
      int f_similarity,
      int f_semisimilarity,
      int *Elt, int verbose_level);
   void matrices_without_eigenvector_one(
      sims *S, int *&Sol, int &cnt,
      int f_path_select, int select_value,
      int verbose_level);
   void matrix_minor(int *Elt, int *Elt1,
      matrix_group *mtx1, int f, int verbose_level);
   int base_len(int verbose_level);
   void base_and_transversal_length(
         int base_len,
         long int *base, int *transversal_length,
         int verbose_level);
   void strong_generators_low_level(int *&data,
         int &size, int &nb_gens, int verbose_level);
   int has_shape_of_singer_cycle(int *Elt);
};




// ##############################################################################
// orbits_on_something.cpp
// ##############################################################################

//! compute orbits of a group in a given action; allows file io

class orbits_on_something {

public:

   actions::action *A;
   strong_generators *SG;
   schreier *Sch;
```

```
int f_load_save;
std::string prefix;
std::string fname;
std::string fname_csv;

data_structures::tally *Classify_orbits_by_length;


orbits_on_something();
~orbits_on_something();
void init(
      actions::action *A,
      strong_generators *SG,
      int f_load_save,
      std::string &prefix,
      int verbose_level);
void stabilizer_any_point(int pt,
      strong_generators *&Stab, int verbose_level);
void stabilizer_of(int orbit_idx, int verbose_level);
void idx_of_points_in_orbits_of_length_l(
      long int *set, int set_sz, int go, int l,
      std::vector<int> &Idx,
      int verbose_level);
void orbit_type_of_set(
      long int *set, int set_sz, int go,
      long int *orbit_type,
      int verbose_level);
// orbit_type[(go + 1) * go] must be allocated beforehand
void report_type(
      std::ostream &ost, long int *orbit_type, long int goi);
void compute_compact_type(
      long int *orbit_type, long int goi,
      long int *&compact_type,
      long int *&row_labels, long int *&col_labels,
      int &m, int &n);
void report_orbit_lengths(std::ostream &ost);
void print_orbits_based_on_filtered_orbits(std::ostream &ost,
      data_structures::set_of_sets *Filtered_orbits);
void classify_orbits_by_length(int verbose_level);
void report_classified_orbit_lengths(std::ostream &ost);
void report_classified_orbits_by_lengths(std::ostream &ost);
int get_orbit_type_index(int orbit_length);
int get_orbit_type_index_if_present(int orbit_length);
void test_all_orbits_by_length(
   int (*test_function)(
         long int *orbit, int orbit_length, void *data),
```

```
   void *test_function_data,
   int verbose_level);
void test_orbits_of_a_certain_length(
   int orbit_length,
   int &type_idx,
   int &prev_nb,
   int (*test_function)(long int *orbit, int orbit_length, void *data),
   void *test_function_data,
   int verbose_level);
void print_orbits_of_a_certain_length(int orbit_length);
int test_pair_of_orbits_of_a_equal_length(
      int orbit_length,
      int type_idx,
      int idx1, int idx2,
      long int *Orbit1,
      long int *Orbit2,
      int (*test_function)(
            long int *orbit1, int orbit_length1,
            long int *orbit2, int orbit_length2, void *data),
      void *test_function_data,
      int verbose_level);
void report_orbits_of_type(std::ostream &ost, int type_idx);
void create_graph_on_orbits_of_a_certain_length_after_filtering(
      graph_theory::colored_graph *&CG,
   std::string &fname,
   long int *filter_by_set,
   int filter_by_set_size,
   int orbit_length,
   int &type_idx,
   int f_has_user_data, long int *user_data, int user_data_size,
   int f_has_colors, int number_colors, int *color_table,
   int (*test_function)(
         long int *orbit1, int orbit_length1,
         long int *orbit2, int orbit_length2, void *data),
   void *test_function_data,
   int verbose_level);
void create_graph_on_orbits_of_a_certain_length(
      graph_theory::colored_graph *&CG,
   std::string &fname,
   int orbit_length,
   int &type_idx,
   int f_has_user_data, long int *user_data, int user_data_size,
   int f_has_colors, int number_colors, int *color_table,
   int (*test_function)(
         long int *orbit1, int orbit_length1,
         long int *orbit2, int orbit_length2, void *data),
   void *test_function_data,
```

```
   int verbose_level);
void extract_orbits(
   int orbit_length,
   int nb_orbits,
   int *orbits,
   long int *extracted_set,
   int verbose_level);
void extract_orbits_using_classification(
   int orbit_length,
   int nb_orbits,
   long int *orbits_idx,
   long int *extracted_set,
   int verbose_level);
void create_graph_on_orbits_of_a_certain_length_override_orbits_classified(
      graph_theory::colored_graph *&CG,
   std::string &fname,
   int orbit_length,
   int &type_idx,
   int f_has_user_data, long int *user_data, int user_data_size,
   int (*test_function)(
         long int *orbit1, int orbit_length1,
         long int *orbit2, int orbit_length2, void *data),
   void *test_function_data,
   data_structures::set_of_sets *my_orbits_classified,
   int verbose_level);
void create_weighted_graph_on_orbits(
      graph_theory::colored_graph *&CG,
   std::string &fname,
   int *Orbit_lengths,
   int nb_orbit_lengths,
   int *&Type_idx,
   int f_has_user_data, long int *user_data, int user_data_size,
   int (*test_function)(
         long int *orbit1, int orbit_length1,
         long int *orbit2, int orbit_length2, void *data),
   void *test_function_data,
   data_structures::set_of_sets *my_orbits_classified,
   int verbose_level);
void compute_orbit_invariant_after_classification(
      data_structures::set_of_sets *&Orbit_invariant,
      int (*evaluate_orbit_invariant_function)(
            int a, int i, int j,
            void *evaluate_data, int verbose_level),
      void *evaluate_data, int verbose_level);
void get_orbit_number_and_position(long int a,
      int &orbit_idx, int &orbit_pos, int verbose_level);
void create_latex_report(int verbose_level);
```

```
   void report(std::ostream &ost, int verbose_level);
   void report_quick(std::ostream &ost, int verbose_level);
   void export_something(std::string &what, int data1,
         std::string &fname, int verbose_level);
   void export_something_worker(
         std::string &fname_base,
         std::string &what, int data1,
         std::string &fname,
         int verbose_level);

};


// ##############################################################################
// permutation_group_create.cpp
// ##############################################################################

//! a domain for permutation groups whose elements are given in the permutation r
epresentation

class permutation_group_create {

public:
   permutation_group_description *Descr;

   std::string label;
   std::string label_tex;

   //strong_generators *initial_strong_gens;
   actions::action *A_initial;

   int f_has_strong_generators;
   strong_generators *Strong_gens;
   actions::action *A2;

   int f_has_nice_gens;
   data_structures_groups::vector_ge *nice_gens;


   permutation_group_create();
   ~permutation_group_create();
   void permutation_group_init(
         permutation_group_description *description,
         int verbose_level);
   void init_subgroup_by_generators(
         std::string &subgroup_label,
         std::string &subgroup_order_text,
```

```
        int nb_subgroup_generators,
        std::string &subgroup_generators_label,
        int verbose_level);


};


// ##############################################################################
// permutation_group_description.cpp
// ##############################################################################

//! a domain for permutation groups whose elements are given in the permutation r
epresentation

class permutation_group_description {

public:
   int degree;
   permutation_group_type type;

   int f_bsgs;
   std::string bsgs_label;
   std::string bsgs_label_tex;
   std::string bsgs_order_text;
   std::string bsgs_base;
   int bsgs_nb_generators;
   std::string bsgs_generators;


   int f_subgroup_by_generators;
   std::string subgroup_label;
   std::string subgroup_order_text;
   int nb_subgroup_generators;
   std::string subgroup_generators_label;

   permutation_group_description();
   ~permutation_group_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();


};
```

```
// ##############################################################################
// permutation_representation_domain.cpp
// ##############################################################################

//! a domain for permutation groups whose elements are given in the permutation r
epresentation

class permutation_representation_domain {

public:
   int degree;

   int f_induced_action;


   int f_product_action;
   int m;
   int n;
   int mn;
   int offset;

   int char_per_elt;
   int elt_size_int;

   int *Elt1, *Elt2, *Elt3, *Elt4;
   uchar *elt1, *elt2, *elt3;
      // temporary storage, used in element_store()
   int *Eltrk1, *Eltrk2, *Eltrk3;
      // used in store / retrieve

   data_structures::page_storage *Elts;

   permutation_representation_domain();
   ~permutation_representation_domain();
   void allocate();
   void init_product_action(int m, int n,
      int page_length_log, int verbose_level);
   void init(int degree, int page_length_log, int verbose_level);
   void init_data(int page_length_log, int verbose_level);
   void init_with_base(int degree,
      int base_length, int *base, int page_length_log,
      actions::action &A, int verbose_level);
   void transversal_rep(int i, int j, int *Elt, int verbose_level);
   void one(int *Elt);
   int is_one(int *Elt);
   void mult(int *A, int *B, int *AB);
   void copy(int *A, int *B);
```

```
   void invert(int *A, int *Ainv);
   void unpack(uchar *elt, int *Elt);
   void pack(int *Elt, uchar *elt);
   void print(int *Elt, std::ostream &ost);
   void print_with_print_point_function(int *Elt,
         std::ostream &ost,
         void (*point_label)(
               std::stringstream &sstr, long int pt, void *data),
         void *point_label_data);
   void code_for_make_element(int *Elt, int *data);
   void print_for_make_element(int *Elt, std::ostream &ost);
   void print_for_make_element_no_commas(int *Elt, std::ostream &ost);
   void print_with_action(actions::action *A, int *Elt, std::ostream &ost);
   void make_element(int *Elt, int *data, int verbose_level);

};


// ##############################################################################
// permutation_representation.cpp
// ##############################################################################

//! homomorphism to a permutation group

class permutation_representation {

public:
   actions::action *A_original;
   int f_stay_in_the_old_action;
   int nb_gens;
   data_structures_groups::vector_ge *gens; // the original generators in action A
_original
   int *Perms; // [nb_gens * degree]
   int degree;
   //longinteger_object target_go;

   permutation_representation_domain *P;
   int perm_offset;
   int elt_size_int;
      // A_original->elt_size_int + P->elt_size_int
   int make_element_size;

   int char_per_elt;
   uchar *elt1; // [char_per_elt]


   std::string label;
```

```
   std::string label_tex;

   data_structures::page_storage *PS;

   int *Elts;
      // [nb_gens * elt_size_int], the generators in the induced action


   permutation_representation();
   ~permutation_representation();
   void init(actions::action *A_original,
         int f_stay_in_the_old_action,
         data_structures_groups::vector_ge *gens,
         int *Perms, int degree,
         int verbose_level);
      // Perms is degree x nb_gens
   long int element_image_of(
         int *Elt, long int a, int verbose_level);
   void element_one(int *Elt);
   int element_is_one(int *Elt);
   void element_mult(int *A, int *B, int *AB, int verbose_level);
   void element_move(int *A, int *B, int verbose_level);
   void element_invert(int *A, int *Av, int verbose_level);
   void element_pack(int *Elt, uchar *elt);
   void element_unpack(uchar *elt, int *Elt);
   void element_print_for_make_element(int *Elt, std::ostream &ost);
   void element_print_easy(int *Elt, std::ostream &ost);
   void element_print_latex(int *Elt, std::ostream &ost);
};



// ##############################################################################
// schreier.cpp
// ##############################################################################

//! Schreier trees for orbits of groups on points

class schreier {

public:
   actions::action *A;
   int f_images_only;
   long int degree;
   data_structures_groups::vector_ge gens;
   data_structures_groups::vector_ge gens_inv;
   int nb_images;
```

```
int **images;
   // [nb_gens][2 * A->degree],
   // allocated by init_images,
   // called from init_generators
   // for each generator,
   // stores the generator as permutation in 0..A->degree-1 ,
   // then the inverse generator in A->degree..2*A->degree-1

int *orbit; // [A->degree]
int *orbit_inv; // [A->degree]

   // prev and label are indexed
   // by the points in the order as listed in orbit.
int *prev; // [A->degree]
int *label; // [A->degree]
//int *orbit_no; // [A->degree]
   // to find out which orbit point a lies in,
   // use orbit_number(pt).
   // It used to be orbit_no[orbit_inv[a]]

int *orbit_first;  // [A->degree + 1]
int *orbit_len;  // [A->degree]
int nb_orbits;

int *Elt1, *Elt2, *Elt3;
int *schreier_gen, *schreier_gen1;
   // used in random_schreier_generator
int *cosetrep, *cosetrep_tmp;
   // used in coset_rep / coset_rep_inv

int f_print_function;
void (*print_function)(std::ostream &ost, int pt, void *data);
void *print_function_data;

int f_preferred_choice_function;
void (*preferred_choice_function)(
      int pt, int &pt_pref, schreier *Sch,
      void *data, int data2, int verbose_level);
void *preferred_choice_function_data;
int preferred_choice_function_data2;

schreier();
~schreier();

schreier(actions::action *A, int verbose_level);
void delete_images();
void init_preferred_choice_function(
```

```
      void (*preferred_choice_function)(
            int pt, int &pt_pref, schreier *Sch,
            void *data, int data2, int verbose_level),
      void *preferred_choice_function_data,
      int preferred_choice_function_data2,
      int verbose_level);
void init_images(int nb_images, int verbose_level);
void init_images_only(int nb_images,
      long int degree, int *images, int verbose_level);
void images_append(int verbose_level);
void init(actions::action *A, int verbose_level);
void allocate_tables();
void init2();
void initialize_tables();
void init_single_generator(int *elt, int verbose_level);
void init_generators(
      data_structures_groups::vector_ge &generators,
      int verbose_level);
void init_images_recycle(int nb_images,
      int **old_images,
      int idx_deleted_generator,
      int verbose_level);
void init_images_recycle(int nb_images,
      int **old_images, int verbose_level);
void init_generators(int nb, int *elt, int verbose_level);
void init_generators_recycle_images(
      data_structures_groups::vector_ge &generators,
      int **old_images,
      int idx_generator_to_delete, int verbose_level);
void init_generators_recycle_images(
      data_structures_groups::vector_ge &generators,
      int **old_images, int verbose_level);


   // elt must point to nb * A->elt_size_in_int
   // int´s that are
   // group elements in int format
void init_generators_recycle_images(int nb,
      int *elt, int **old_images,
      int idx_generator_to_delete, int verbose_level);
void init_generators_recycle_images(int nb,
      int *elt, int **old_images, int verbose_level);
void init_generators_by_hdl(int nb_gen, int *gen_hdl,
   int verbose_level);
void init_generators_by_handle(
      std::vector<int> &gen_hdl,
      int verbose_level);
```

```
long int get_image(
      long int i, int gen_idx, int verbose_level);
void swap_points(int i, int j, int verbose_level);
void move_point_here(int here, int pt);
int orbit_representative(int pt);
int depth_in_tree(int j);
   // j is a coset, not a point
void transporter_from_orbit_rep_to_point(int pt,
   int &orbit_idx, int *Elt, int verbose_level);
void transporter_from_point_to_orbit_rep(int pt,
   int &orbit_idx, int *Elt, int verbose_level);
void coset_rep(int j, int verbose_level);
   // j is a coset, not a point
   // result is in cosetrep
   // determines an element in the group
   // that moves the orbit representative
   // to the j-th point in the orbit.
void coset_rep_with_verbosity(int j, int verbose_level);
void coset_rep_inv(int j, int verbose_level);
void extend_orbit(int *elt, int verbose_level);
void compute_all_point_orbits(int verbose_level);
void compute_all_point_orbits_with_prefered_reps(
   int *prefered_reps, int nb_prefered_reps,
   int verbose_level);
void compute_all_point_orbits_with_preferred_labels(
   long int *preferred_labels, int verbose_level);
void compute_all_orbits_on_invariant_subset(int len,
   long int *subset, int verbose_level);
void compute_all_orbits_on_invariant_subset_lint(
   int len, long int *subset, int verbose_level);
void compute_point_orbit(int pt, int verbose_level);
void compute_point_orbit_with_limited_depth(
      int pt, int max_depth, int verbose_level);
int sum_up_orbit_lengths();
void non_trivial_random_schreier_generator(
      actions::action *A_original,
      int *Elt, int verbose_level);
   // computes non trivial random Schreier
   // generator into schreier_gen
   // non-trivial is with respect to A_original
void random_schreier_generator_ith_orbit(
      int *Elt, int orbit_no,
      int verbose_level);
   // computes random Schreier generator
   // for the orbit orbit_no into Elt
void random_schreier_generator(
      int *Elt, int verbose_level);
```

```
   // computes random Schreier generator
   // for the first orbit into Elt
void trace_back(int *path, int i, int &j);
void intersection_vector(int *set, int len,
   int *intersection_cnt);
void orbits_on_invariant_subset_fast(int len,
   int *subset, int verbose_level);
void orbits_on_invariant_subset_fast_lint(
   int len, long int *subset, int verbose_level);
void orbits_on_invariant_subset(
      int len, int *subset,
   int &nb_orbits_on_subset,
   int *&orbit_perm, int *&orbit_perm_inv);
void get_orbit_partition_of_points_and_lines(
      data_structures::partitionstack &S,
      int verbose_level);
void get_orbit_partition(
      data_structures::partitionstack &S,
   int verbose_level);
void get_orbit_in_order(std::vector<int> &Orb,
   int orbit_idx, int verbose_level);
strong_generators *stabilizer_any_point_plus_cosets(
      actions::action *default_action,
   ring_theory::longinteger_object &full_group_order,
   int pt, data_structures_groups::vector_ge *&cosets,
   int verbose_level);
strong_generators *stabilizer_any_point(
      actions::action *default_action,
   ring_theory::longinteger_object &full_group_order,
   int pt,
   int verbose_level);
data_structures_groups::set_and_stabilizer
   *get_orbit_rep(
         actions::action *default_action,
      ring_theory::longinteger_object &full_group_order,
      int orbit_idx, int verbose_level);
void get_orbit_rep_to(actions::action *default_action,
      ring_theory::longinteger_object &full_group_order,
      int orbit_idx,
      data_structures_groups::set_and_stabilizer *Rep,
      int verbose_level);
strong_generators *stabilizer_orbit_rep(
      actions::action *default_action,
   ring_theory::longinteger_object &full_group_order,
   int orbit_idx, int verbose_level);
void point_stabilizer(
      actions::action *default_action,
```

```
      ring_theory::longinteger_object &go,
      groups::sims *&Stab,
      int orbit_no, int verbose_level);
   // this function allocates a sims structure into Stab.
void get_orbit(int orbit_idx, long int *set, int &len,
   int verbose_level);
void compute_orbit_statistic(int *set, int set_size,
   int *orbit_count, int verbose_level);
void compute_orbit_statistic_lint(
      long int *set, int set_size,
   int *orbit_count, int verbose_level);
void orbits_as_set_of_sets(
      data_structures::set_of_sets *&S,
      int verbose_level);
void get_orbit_reps(
      int *&Reps, int &nb_reps, int verbose_level);
int find_shortest_orbit_if_unique(int &idx);
void elements_in_orbit_of(int pt, int *orb, int &nb,
   int verbose_level);
void get_orbit_length(
      int *&orbit_length, int verbose_level);
void get_orbit_lengths_once_each(
      int *&orbit_lengths,
   int &nb_orbit_lengths);
int orbit_number(int pt);
void get_orbit_number_and_position(
      int pt, int &orbit_idx, int &orbit_pos,
      int verbose_level);
void get_orbit_decomposition_scheme_of_graph(
   int *Adj, int n, int *&Decomp_scheme, int verbose_level);
void create_point_list_sorted(
      int *&point_list, int &point_list_length);
void shallow_tree_generators(int orbit_idx,
      int f_randomized,
      schreier *&shallow_tree,
      int verbose_level);
data_structures_groups::schreier_vector *get_schreier_vector(
      int gen_hdl_first, int nb_gen,
      enum shallow_schreier_tree_strategy
         Shallow_schreier_tree_strategy,
      int verbose_level);
int get_num_points();
   // This function returns the number of points in the
   // schreier forest
double get_average_word_length();
   // This function returns the average word length of the forest.
double get_average_word_length(int orbit_idx);
```

```
void compute_orbit_invariant(int *&orbit_invariant,
      int (*compute_orbit_invariant_callback)(schreier *Sch,
            int orbit_idx, void *data, int verbose_level),
      void *compute_orbit_invariant_data,
      int verbose_level);
void print_TDA(
      std::ostream &ost,
      geometry::object_with_canonical_form *OwCF,
      combinatorics::classification_of_objects_report_options
         *Report_options,
      int verbose_level);
void latex_TDA(std::ostream &ost,
      combinatorics::encoded_combinatorial_object *Enc,
      int verbose_level);

// schreier_io.cpp:
void latex(std::string &fname);
void print_orbit_lengths(std::ostream &ost);
void print_orbit_lengths_tex(std::ostream &ost);
void print_fixed_points_tex(std::ostream &ost);
void print_orbit_length_distribution(
      std::ostream &ost);
void print_orbit_length_distribution_to_string(
      std::string &str);
void print_orbit_reps(std::ostream &ost);
void print(std::ostream &ost);
void print_and_list_orbits(std::ostream &ost);
void print_and_list_orbits_with_original_labels(std::ostream &ost);
void print_and_list_orbits_tex(std::ostream &ost);
void print_and_list_non_trivial_orbits_tex(std::ostream &ost);
void print_and_list_all_orbits_and_stabilizers_with_list_of_elements_tex(
      std::ostream &ost,
      actions::action *default_action,
      strong_generators *gens,
      int verbose_level);
void make_orbit_trees(std::ostream &ost,
      std::string &fname_mask,
      graphics::layered_graph_draw_options *Opt,
      int verbose_level);
void print_and_list_orbits_with_original_labels_tex(
      std::ostream &ost);
void print_and_list_orbits_of_given_length(std::ostream &ost,
   int len);
void print_and_list_orbits_and_stabilizer(
      std::ostream &ost,
      actions::action *default_action,
      ring_theory::longinteger_object &go,
```

```
   void (*print_point)(
         std::ostream &ost, int pt, void *data),
      void *data);
void print_and_list_orbits_using_labels(std::ostream &ost,
   long int *labels);
void print_tables(std::ostream &ost, int f_with_cosetrep);
void print_tables_latex(
      std::ostream &ost, int f_with_cosetrep);
void print_generators();
void print_generators_latex(
      std::ostream &ost);
void print_generators_with_permutations();
void print_orbit(int orbit_no);
void print_orbit_using_labels(
      int orbit_no, long int *labels);
void print_orbit(std::ostream &ost, int orbit_no);
void print_orbit_with_original_labels(
      std::ostream &ost, int orbit_no);
void print_orbit_tex(std::ostream &ost, int orbit_no);
void print_orbit_sorted_tex(
      std::ostream &ost,
      int orbit_no, int f_truncate, int max_length);
void get_orbit_sorted(int *&v, int &len, int orbit_no);
void print_and_list_orbit_and_stabilizer_tex(int i,
      actions::action *default_action,
   ring_theory::longinteger_object &full_group_order,
   std::ostream &ost);
void write_orbit_summary(std::string &fname,
      actions::action *default_action,
      ring_theory::longinteger_object &full_group_order,
      int verbose_level);
void print_and_list_orbit_and_stabilizer_with_list_of_elements_tex(
   int i, actions::action *default_action,
   strong_generators *gens, std::ostream &ost);
void print_and_list_orbit_tex(int i, std::ostream &ost);
void print_and_list_orbits_sorted_by_length_tex(
      std::ostream &ost);
void print_and_list_orbits_and_stabilizer_sorted_by_length(
      std::ostream &ost, int f_tex,
      actions::action *default_action,
   ring_theory::longinteger_object &full_group_order);
void print_fancy(
      std::ostream &ost, int f_tex,
      actions::action *default_action,
      strong_generators *gens_full_group);
void print_and_list_orbits_sorted_by_length(
      std::ostream &ost);
```

```
void print_and_list_orbits_sorted_by_length(
      std::ostream &ost, int f_tex);
void print_orbit_sorted_with_original_labels_tex(
      std::ostream &ost,
      int orbit_no, int f_truncate, int max_length);
void print_orbit_using_labels(
      std::ostream &ost, int orbit_no, long int *labels);
void print_orbit_using_callback(
      std::ostream &ost, int orbit_no,
   void (*print_point)(
         std::ostream &ost, int pt, void *data),
   void *data);
void print_orbit_type(int f_backwards);
void list_all_orbits_tex(std::ostream &ost);
void print_orbit_through_labels(std::ostream &ost,
   int orbit_no, long int *point_labels);
void print_orbit_sorted(
      std::ostream &ost, int orbit_no);
void print_orbit(int cur, int last);
void print_tree(int orbit_no);
void export_tree_as_layered_graph(int orbit_no,
      std::string &fname_mask,
      int verbose_level);
void draw_forest(std::string &fname_mask,
      graphics::layered_graph_draw_options *Opt,
      int f_has_point_labels, long int *point_labels,
      int verbose_level);
void draw_tree(std::string &fname,
      graphics::layered_graph_draw_options *Opt,
      int orbit_no,
      int f_has_point_labels, long int *point_labels,
      int verbose_level);
void draw_tree2(std::string &fname,
      graphics::layered_graph_draw_options *Opt,
      int *weight, int *placement_x, int max_depth,
      int i, int last,
      int f_has_point_labels, long int *point_labels,
      int verbose_level);
void subtree_draw_lines(
      graphics::mp_graphics &G,
      graphics::layered_graph_draw_options *Opt,
      int parent_x, int parent_y, int *weight,
      int *placement_x, int max_depth, int i, int last,
      int y_max,
      int verbose_level);
void subtree_draw_vertices(
      graphics::mp_graphics &G,
```

```
        graphics::layered_graph_draw_options *Opt,
        int parent_x, int parent_y, int *weight,
        int *placement_x, int max_depth, int i, int last,
        int f_has_point_labels, long int *point_labels,
        int y_max,
        int verbose_level);
   void subtree_place(int *weight, int *placement_x,
      int left, int right, int i, int last);
   int subtree_calc_weight(int *weight, int &max_depth,
      int i, int last);
   int subtree_depth_first(
        std::ostream &ost, int *path, int i, int last);
   void print_path(std::ostream &ost, int *path, int l);
   void write_to_file_csv(std::string &fname_csv, int verbose_level);
   void write_to_file_binary(std::ofstream &fp, int verbose_level);
   void read_from_file_binary(std::ifstream &fp, int verbose_level);
   void write_file_binary(std::string &fname, int verbose_level);
   void read_file_binary(std::string &fname, int verbose_level);
   void list_elements_as_permutations_vertically(std::ostream &ost);
};

// ##############################################################################
// schreier_sims.cpp
// ##############################################################################


//! Schreier Sims algorithm to create the stabilizer chain of a permutation group


class schreier_sims {

public:
   actions::action *GA;
   sims *G;

   int f_interested_in_kernel;
   actions::action *KA;
   sims *K;

   ring_theory::longinteger_object G_order, K_order, KG_order;

   int *Elt1;
   int *Elt2;
   int *Elt3;

   int f_has_target_group_order;
   ring_theory::longinteger_object tgo; // target group order
```

```
int f_from_generators;
data_structures_groups::vector_ge *external_gens;

int f_from_random_process;
void (*callback_choose_random_generator)(int iteration,
   int *Elt, void *data, int verbose_level);
void *callback_choose_random_generator_data;

int f_from_old_G;
sims *old_G;

int f_has_base_of_choice;
int base_of_choice_len;
int *base_of_choice;

int f_override_choose_next_base_point_method;
int (*choose_next_base_point_method)(actions::action *A,
   int *Elt, int verbose_level);

int iteration;

schreier_sims();
~schreier_sims();
void init(actions::action *A, int verbose_level);
void interested_in_kernel(actions::action *KA, int verbose_level);
void init_target_group_order(
      ring_theory::longinteger_object &tgo,
   int verbose_level);
void init_generators(
      data_structures_groups::vector_ge *gens, int verbose_level);
void init_random_process(
   void (*callback_choose_random_generator)(
   int iteration, int *Elt, void *data,
   int verbose_level),
   void *callback_choose_random_generator_data,
   int verbose_level);
void init_old_G(sims *old_G, int verbose_level);
void init_base_of_choice(
   int base_of_choice_len, int *base_of_choice,
   int verbose_level);
void init_choose_next_base_point_method(
   int (*choose_next_base_point_method)(actions::action *A,
   int *Elt, int verbose_level),
   int verbose_level);
void compute_group_orders();
```

```
   void print_group_orders();
   void get_generator_internal(int *Elt, int verbose_level);
   void get_generator_external(int *Elt, int verbose_level);
   void get_generator_external_from_generators(int *Elt,
      int verbose_level);
   void get_generator_external_random_process(int *Elt,
      int verbose_level);
   void get_generator_external_old_G(int *Elt,
      int verbose_level);
   void get_generator(int *Elt, int verbose_level);
   void closure_group(int verbose_level);
   void create_group(int verbose_level);
};

// ##############################################################################
// sims.cpp
// ##############################################################################

//! a permutation group represented via a stabilizer chain

class sims {

public:
   actions::action *A;

   int my_base_len;

   data_structures_groups::vector_ge gens;
   data_structures_groups::vector_ge gens_inv;

   int *gen_depth; // [nb_gen]
   int *gen_perm; // [nb_gen]

   int *nb_gen; // [my_base_len + 1]
      // nb_gen[i] is the number of generators
      // which stabilize the base points 0,...,i-1,
      // i.e. which belong to G^{(i)}.
      // The actual generator index ("gen_idx") must be obtained
      // from the array gen_perm[].
      // Thus, gen_perm[j] for 0 \le j < nb_gen[i] are the
      // indices of generators which belong to G^{(i)}
      // the generators for G^{(i)} modulo G^{(i+1)}
      // those indexed by nb_gen[i + 1], .., nb_gen[i] - 1 (!!!)
      // Observe that the entries in nb_gen[] are *decreasing*.
      // This is because the generators at the bottom of the
      // stabilizer chain are listed first.
      // (And nb_gen[0] is the total number of generators).
```

```
   int transversal_length;
      // an upper bound for the length of every basic orbit

   int *path; // [my_base_len]

   int nb_images;
   int **images;


private:
   // stabilizer chain:

   int *orbit_len; // [my_base_len]
      // orbit_len[i] is the length of the i-th basic orbit.

   int **orbit;
      // [my_base_len][transversal_length]
      // orbit[i][j] is the j-th point in the orbit
      // of the i-th base point.
      // for 0 \le j < orbit_len[i].
      // for orbit_len[i] \le j < A->deg,
      // the points not in the orbit are listed.
   int **orbit_inv;
      // [my_base_len][transversal_length]
      // orbit[i] is the inverse of the permutation orbit[i],
      // i.e. given a point j,
      // orbit_inv[i][j] is the coset (or position in the orbit)
      // which contains j.

   int **prev; // [my_base_len][transversal_length]
   int **label; // [my_base_len][transversal_length]


   // this is wrong, Path and Label describe a path in a schreier tree
   // and hence should be allocated according
   // to the largest degree, not the base length
   //int *Path; // [my_base_len + 1]
   //int *Label; // [my_base_len]


   // storage for temporary data and
   // group elements computed by various routines.
   int *Elt1, *Elt2, *Elt3, *Elt4;
   int *strip1, *strip2;
      // used in strip
```

```
   int *eltrk1, *eltrk2, *eltrk3;
      // used in element rank unrank
   int *cosetrep_tmp;
      // used in coset_rep / coset_rep_inv
   int *schreier_gen, *schreier_gen1;
      // used in random_schreier_generator

   int *cosetrep;
public:


   // sims.cpp:
   sims();
   ~sims();
   sims(actions::action *A, int verbose_level);

   void delete_images();
   void init_images(int nb_images);
   void images_append();
   void init(actions::action *A, int verbose_level);
      // initializes the trivial group
      // with the base as given in A
   void init_cyclic_group_from_generator(actions::action *A,
         int *Elt, int verbose_level);
   // initializes the cyclic group generated by Elt with the base as given in A
   void init_without_base(
         actions::action *A, int verbose_level);
   void reallocate_base(int old_base_len, int verbose_level);
   void initialize_table(int i, int verbose_level);
   void init_trivial_group(int verbose_level);
      // clears the generators array,
      // and sets the i-th transversal to contain
      // only the i-th base point (for all i).
   void init_trivial_orbit(int i, int verbose_level);
   void init_generators(
         data_structures_groups::vector_ge &generators,
         int verbose_level);
   void init_generators(int nb, int *elt, int verbose_level);
      // copies the given elements into the generator array,
      // then computes depth and perm
   void init_generators_by_hdl(
         int nb_gen, int *gen_hdl, int verbose_level);
   void init_generator_depth_and_perm(int verbose_level);
   void add_generator(int *elt, int verbose_level);
      // adds elt to list of generators,
      // computes the depth of the element,
      // updates the arrays gen_depth and gen_perm accordingly
```

```
   // does not change the transversals
int generator_depth(int gen_idx);
   // returns the index of the first base point
   // which is moved by a given generator.
int generator_depth(int *elt);
   // returns the index of the first base point
   // which is moved by the given element
void group_order(ring_theory::longinteger_object &go);
void group_order_verbose(ring_theory::longinteger_object &go,
      int verbose_level);
void subgroup_order_verbose(ring_theory::longinteger_object &go,
      int level, int verbose_level);
long int group_order_lint();
int is_trivial_group();
int last_moved_base_point();
   // j == -1 means the group is trivial
int get_image(int i, int gen_idx);
   // get the image of a point i under generator gen_idx,
   // goes through a
   // table of stored images by default.
   // Computes the image only if not yet available.
int get_image(int i, int *elt);
   // get the image of a point i under a given group element,
   // does not go through a table.
void swap_points(int lvl, int i, int j);
   // swaps two points given by their cosets
void path_unrank_lint(long int a);
long int path_rank_lint();

void element_from_path(int *elt, int verbose_level);
   // given coset representatives in path[],
   // the corresponding
   // element is multiplied.
   // uses eltrk1, eltrk2
void element_from_path_inv(int *elt);
void element_unrank(
      ring_theory::longinteger_object &a, int *elt,
   int verbose_level);
void element_unrank(
      ring_theory::longinteger_object &a, int *elt);
   // Returns group element whose rank is a.
   // the elements represented by the chain are
   // enumerated 0, ... go - 1
   // with the convention that 0 always stands
   // for the identity element.
   // The computed group element will be computed into Elt1
void element_rank(
```

```
      ring_theory::longinteger_object &a, int *elt);
   // Computes the rank of the element in elt into a.
   // uses eltrk1, eltrk2
void element_unrank_lint(long int rk, int *Elt, int verbose_level);
void element_unrank_lint(long int rk, int *Elt);
long int element_rank_lint(int *Elt);
int is_element_of(int *elt, int verbose_level);
void test_element_rank_unrank();
void coset_rep(int *Elt, int i, int j, int verbose_level);
   // computes a coset representative in transversal i
   // which maps
   // the i-th base point to the point which is in
   // coset j of the i-th basic orbit.
   // j is a coset, not a point
   // result is in cosetrep
int compute_coset_rep_depth(int i, int j, int verbose_level);
void compute_coset_rep_path(int i, int j, int &depth,
      int *&Path, int *&Label,
   int verbose_level);
void coset_rep_inv(int *Elt, int i, int j, int verbose_level_le);
   // computes the inverse element of what coset_rep computes,
   // i.e. an element which maps the
   // j-th point in the orbit to the
   // i-th base point.
   // j is a coset, not a point
   // result is in cosetrep
void extract_strong_generators_in_order(
      data_structures_groups::vector_ge &SG,
   int *tl, int verbose_level);
void random_schreier_generator(int *Elt, int verbose_level);
   // computes random Schreier generator
void element_as_permutation(actions::action *A_special,
   long int elt_rk, int *perm, int verbose_level);
int least_moved_point_at_level(int lvl, int verbose_level);
int get_orbit(int i, int j);
int get_orbit_inv(int i, int j);
int get_orbit_length(int i);
void get_orbit(
      int orbit_idx, std::vector<int> &Orb,
      int verbose_level);
void all_elements(
      data_structures_groups::vector_ge *&vec,
      int verbose_level);
void all_elements_save_csv(std::string &fname, int verbose_level);
void all_elements_export_inversion_graphs(
      std::string &fname, int verbose_level);
```

```
// sims_main.cpp:
void compute_base_orbits(int verbose_level);
void compute_base_orbits_known_length(int *tl,
   int verbose_level);
void extend_base_orbit(int new_gen_idx, int lvl,
   int verbose_level);
void compute_base_orbit(int lvl, int verbose_level);
   // applies all generators at the given level to compute
   // the corresponding basic orbit.
   // the generators are the first nb_gen[lvl]
   // in the generator array
void compute_base_orbit_known_length(int lvl,
   int target_length, int verbose_level);
int strip_and_add(int *elt, int *residue, int verbose_level);
   // returns TRUE if something was added,
   // FALSE if element stripped through
int strip(int *elt, int *residue, int &drop_out_level,
   int &image, int verbose_level);
   // returns TRUE if the element sifts through
void add_generator_at_level(int *elt, int lvl,
   int verbose_level);
   // add the generator to the array of generators
   // and then extends the
   // basic orbits 0,..,lvl using extend_base_orbit
void add_generator_at_level_only(int *elt,
   int lvl, int verbose_level);
   // add the generator to the array of generators
   // and then extends the
   // basic orbit lvl using extend_base_orbit
void build_up_group_random_process_no_kernel(sims *old_G,
   int verbose_level);
void extend_group_random_process_no_kernel(
      sims *extending_by_G,
      ring_theory::longinteger_object &target_go,
   int verbose_level);
void build_up_group_random_process(sims *K, sims *old_G,
      ring_theory::longinteger_object &target_go,
   int f_override_choose_next_base_point,
   int (*choose_next_base_point_method)(actions::action *A,
      int *Elt, int verbose_level),
   int verbose_level);
void build_up_group_from_generators(sims *K,
      data_structures_groups::vector_ge *gens,
   int f_target_go, ring_theory::longinteger_object *target_go,
   int f_override_choose_next_base_point,
   int (*choose_next_base_point_method)(actions::action *A,
```

```
      int *Elt, int verbose_level),
   int verbose_level);
int closure_group(int nb_times, int verbose_level);




// sims2.cpp
void build_up_subgroup_random_process(sims *G,
   void (*choose_random_generator_for_subgroup)(
      sims *G, int *Elt, int verbose_level),
   int verbose_level);

// sims3.cpp
void subgroup_make_characteristic_vector(sims *Sub,
   int *C, int verbose_level);
void normalizer_based_on_characteristic_vector(int *C_sub,
   int *Gen_idx, int nb_gens, int *N, long int &N_go,
   int verbose_level);
void order_structure_relative_to_subgroup(int *C_sub,
   int *Order, int *Residue, int verbose_level);



// sims_group_theory.cpp:
void random_element(int *elt, int verbose_level);
   // compute a random element among the group elements
   // represented by the chain
   // (chooses random cosets along the stabilizer chain)
void random_element_of_order(int *elt, int order,
   int verbose_level);
void random_elements_of_order(data_structures_groups::vector_ge *elts,
   int *orders, int nb, int verbose_level);
void transitive_extension(schreier &O,
      data_structures_groups::vector_ge &SG,
   int *tl, int verbose_level);
int transitive_extension_tolerant(schreier &O,
      data_structures_groups::vector_ge &SG, int *tl, int f_tolerant,
   int verbose_level);
void transitive_extension_using_coset_representatives_extract_generators(
   int *coset_reps, int nb_cosets,
   data_structures_groups::vector_ge &SG, int *tl,
   int verbose_level);
void transitive_extension_using_coset_representatives(
   int *coset_reps, int nb_cosets,
   int verbose_level);
void transitive_extension_using_generators(
```

```
   int *Elt_gens, int nb_gens, int subgroup_index,
   data_structures_groups::vector_ge &SG, int *tl,
   int verbose_level);
void point_stabilizer_stabchain_with_action(
      actions::action *A2,
   sims &S, int pt, int verbose_level);
   // first computes the orbit of the point pt
   // in action A2 under the generators
   // that are stored at present
   // (using a temporary schreier object),
   // then sifts random schreier generators into S
void point_stabilizer(data_structures_groups::vector_ge &SG, int *tl,
   int pt, int verbose_level);
   // computes strong generating set
   // for the stabilizer of point pt
void point_stabilizer_with_action(actions::action *A2,
      data_structures_groups::vector_ge &SG,
      int *tl, int pt,
      int verbose_level);
   // computes strong generating set for
   // the stabilizer of point pt in action A2
void conjugate(actions::action *A, sims *old_G, int *Elt,
   int f_overshooting_OK, int verbose_level);
   // Elt * g * Elt^{-1} where g is in old_G
int test_if_in_set_stabilizer(actions::action *A,
   long int *set, int size, int verbose_level);
int test_if_subgroup(sims *old_G, int verbose_level);
int find_element_with_exactly_n_fixpoints_in_given_action(
      int *Elt, int nb_fixpoints,
      actions::action *A_given, int verbose_level);
void table_of_group_elements_in_data_form(int *&Table,
   int &len, int &sz, int verbose_level);
void regular_representation(int *Elt, int *perm,
   int verbose_level);
void center(data_structures_groups::vector_ge &gens,
      int *center_element_ranks, int &nb_elements,
      int verbose_level);
void all_cosets(int *subset, int size,
   long int *all_cosets, int verbose_level);
void element_ranks_subgroup(sims *subgroup,
   int *element_ranks, int verbose_level);
void find_standard_generators_int(int ord_a, int ord_b,
   int ord_ab, int &a, int &b, int &nb_trials,
   int verbose_level);
long int find_element_of_given_order_int(int ord,
   int &nb_trials, int verbose_level);
int find_element_of_given_order_int(int *Elt,
```

```
   int ord, int &nb_trials, int max_trials,
   int verbose_level);
void find_element_of_prime_power_order(int p,
   int *Elt, int &e, int &nb_trials, int verbose_level);
void evaluate_word_int(int word_len,
   int *word, int *Elt, int verbose_level);
void sylow_subgroup(int p, sims *P, int verbose_level);
int is_normalizing(int *Elt, int verbose_level);
void create_Cayley_graph(data_structures_groups::vector_ge *gens,
      int *&Adj, long int &n,
   int verbose_level);
void create_group_table(int *&Table, long int &n, int verbose_level);
void compute_conjugacy_classes(
      actions::action *&Aconj,
      induced_actions::action_by_conjugation *&ABC, schreier *&Sch,
   strong_generators *&SG, int &nb_classes,
   int *&class_size, int *&class_rep,
   int verbose_level);
void compute_all_powers(int elt_idx, int n, int *power_elt,
      int verbose_level);
long int mult_by_rank(long int rk_a, long int rk_b, int verbose_level);
long int mult_by_rank(long int rk_a, long int rk_b);
long int invert_by_rank(long int rk_a, int verbose_level);
long int conjugate_by_rank(
      long int rk_a, long int rk_b, int verbose_level);
   // computes b^{-1} * a * b
long int conjugate_by_rank_b_bv_given(long int rk_a,
   int *Elt_b, int *Elt_bv, int verbose_level);
void zuppo_list(
      int *Zuppos, int &nb_zuppos, int verbose_level);
void dimino(
   int *subgroup, int subgroup_sz, int *gens, int &nb_gens,
   int *cosets,
   int new_gen,
   int *group, int &group_sz,
   int verbose_level);
void Cayley_graph(int *&Adj, int &sz,
      data_structures_groups::vector_ge *gens_S,
   int verbose_level);


// sims_io.cpp:
void create_group_tree(std::string &fname,
      int f_full, int verbose_level);
void print_transversals();
void print_transversals_short();
void print_transversal_lengths();
```

```
   void print_orbit_len();
   void print(int verbose_level);
   void print_generators();
   void print_generators_tex(std::ostream &ost);
   void print_generators_as_permutations();
   void print_generators_as_permutations_override_action(
         actions::action *A);
   void print_basic_orbits();
   void print_basic_orbit(int i);
   void print_generator_depth_and_perm();
   void print_group_order(std::ostream &ost);
   void print_group_order_factored(std::ostream &ost);
   void print_generators_at_level_or_below(int lvl);
   void write_all_group_elements(
         std::string &fname, int verbose_level);
   void print_all_group_elements_to_file(std::string &fname,
      int verbose_level);
   void print_all_group_elements();
   void print_all_group_elements_tex(std::ostream &ost,
         int f_with_permutation,
         int f_override_action, actions::action *A_special);
   void print_all_group_elements_tree(std::ostream &ost);
   void print_all_group_elements_with_permutations_tex(
         std::ostream &ost);
   void print_all_group_elements_as_permutations();
   void print_all_group_elements_as_permutations_in_special_action(
         actions::action *A_special);
   void print_all_transversal_elements();
   void save_list_of_elements(std::string &fname,
      int verbose_level);
   void read_list_of_elements(actions::action *A,
         std::string &fname, int verbose_level);
   void report(std::ostream &ost,
         std::string &prefix,
         graphics::layered_graph_draw_options *LG_Draw_options,
         int verbose_level);


};

// sims2.cpp
void choose_random_generator_derived_group(sims *G, int *Elt,
   int verbose_level);
```

```
// ##############################################################################
// strong_generators.cpp
// ##############################################################################

//! a strong generating set for a permutation group with respect to a fixed actio
n

class strong_generators {
public:

   actions::action *A;
   int *tl;
   data_structures_groups::vector_ge *gens;

   strong_generators();
   ~strong_generators();
   void swap_with(strong_generators *SG);
   void init(actions::action *A);
   void init(actions::action *A, int verbose_level);
   void init_from_sims(groups::sims *S, int verbose_level);
   void init_from_ascii_coding(actions::action *A,
         std::string &ascii_coding, int verbose_level);
   strong_generators *create_copy(int verbose_level);
   void init_copy(strong_generators *S,
      int verbose_level);
   void init_by_hdl_and_with_tl(actions::action *A,
         std::vector<int> &gen_handle,
         std::vector<int> &tl,
         int verbose_level);
   void init_by_hdl(actions::action *A, int *gen_hdl,
      int nb_gen, int verbose_level);
   void init_from_permutation_representation(
         actions::action *A,
         sims *parent_group_S, int *data,
         int nb_elements, long int group_order,
         data_structures_groups::vector_ge *&nice_gens,
         int verbose_level);
   void init_from_data(
         actions::action *A, int *data,
      int nb_elements, int elt_size,
      int *transversal_length,
      data_structures_groups::vector_ge *&nice_gens,
      int verbose_level);
   void init_from_data_with_target_go_ascii(
         actions::action *A,
      int *data,
```

```
   int nb_elements, int elt_size,
   std::string &ascii_target_go,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);
void init_from_data_with_target_go(
      actions::action *A,
   int *data_gens,
   int data_gens_size, int nb_gens,
   ring_theory::longinteger_object &target_go,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);
void init_from_data_with_go(
      actions::action *A, std::string &generators_data,
   std::string &go_text,
   int verbose_level);
void init_point_stabilizer_of_arbitrary_point_through_schreier(
   schreier *Sch,
   int pt, int &orbit_idx,
   ring_theory::longinteger_object &full_group_order,
   int verbose_level);
void init_point_stabilizer_orbit_rep_schreier(
      schreier *Sch,
   int orbit_idx,
   ring_theory::longinteger_object &full_group_order,
   int verbose_level);
void init_generators_for_the_conjugate_group_avGa(
   strong_generators *SG, int *Elt_a, int verbose_level);
void init_generators_for_the_conjugate_group_aGav(
   strong_generators *SG, int *Elt_a, int verbose_level);
void init_transposed_group(strong_generators *SG,
   int verbose_level);
void init_group_extension(strong_generators *subgroup,
   int *data, int index,
   int verbose_level);
void init_group_extension(strong_generators *subgroup,
      data_structures_groups::vector_ge *new_gens, int index,
   int verbose_level);
void switch_to_subgroup(
      std::string &rank_vector_text,
      std::string &subgroup_order_text, sims *S,
   int *&subgroup_gens_idx, int &nb_subgroup_gens,
   int verbose_level);
void init_subgroup(actions::action *A, int *subgroup_gens_idx,
   int nb_subgroup_gens,
   const char *subgroup_order_text,
   sims *S,
   int verbose_level);
```

```
void init_subgroup_by_generators(actions::action *A,
   int nb_subgroup_gens,
   int *subgroup_gens,
   std::string &subgroup_order_text,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);
sims *create_sims(int verbose_level);
sims *create_sims_in_different_action(actions::action *A_given,
   int verbose_level);
void add_generators(
      data_structures_groups::vector_ge *coset_reps,
   int group_index, int verbose_level);
void add_single_generator(int *Elt,
   int group_index, int verbose_level);
void group_order(ring_theory::longinteger_object &go);
long int group_order_as_lint();
void print_group_order(std::ostream &ost);
void print_generators_in_source_code();
void print_generators_in_source_code_to_file(std::string &fname);
void print_generators_even_odd();
void print_generators_MAGMA(
      actions::action *A, std::ostream &ost);
void export_magma(
      actions::action *A, std::ostream &ost,
      int verbose_level);
void export_fining(
      actions::action *A, std::ostream &ost,
      int verbose_level);
// at the moment, A is not used
void canonical_image_GAP(
      std::string &input_set_text, std::ostream &ost,
      int verbose_level);
void print_generators_gap(std::ostream &ost);
void print_generators_gap_in_different_action(
      std::ostream &ost, actions::action *A2);
void print_generators_compact(std::ostream &ost);
void print_generators(std::ostream &ost);
void print_generators_in_latex_individually(std::ostream &ost);
void print_generators_tex();
void print_generators_tex(std::ostream &ost);
void print_for_make_element(std::ostream &ost);
void print_generators_in_different_action_tex(
      std::ostream &ost, actions::action *A2);
void print_generators_tex_with_print_point_function(
      actions::action *A,
      std::ostream &ost,
      void (*point_label)(
```

```
            std::stringstream &sstr, long int pt, void *data),
      void *point_label_data);
void print_generators_for_make_element(std::ostream &ost);
void print_generators_as_permutations();
void print_generators_as_permutations_tex(
      std::ostream &ost, actions::action *A2);
void print_with_given_action(std::ostream &ost, actions::action *A2);
void print_elements_ost(std::ostream &ost);
void print_elements_with_special_orthogonal_action_ost(std::ostream &ost);
void print_elements_with_given_action(
      std::ostream &ost, actions::action *A2);
void print_elements_latex_ost(std::ostream &ost);
void print_elements_latex_ost_with_print_point_function(
      actions::action *A,
      std::ostream &ost,
      void (*point_label)(std::stringstream &sstr, long int pt, void *data),
      void *point_label_data);
void create_group_table(int *&Table, long int &go,
   int verbose_level);
void list_of_elements_of_subgroup(
   strong_generators *gens_subgroup,
   long int *&Subgroup_elements_by_index,
   long int &sz_subgroup, int verbose_level);
void compute_schreier_with_given_action(actions::action *A_given,
   schreier *&Sch, int verbose_level);
void compute_schreier_with_given_action_on_a_given_set(
      actions::action *A_given,
   schreier *&Sch, long int *set, int len, int verbose_level);
void orbits_on_points(int &nb_orbits, int *&orbit_reps,
   int verbose_level);
void orbits_on_set_with_given_action_after_restriction(
      actions::action *A_given, long int *Set, int set_sz,
      std::stringstream &orbit_type,
      int verbose_level);
void extract_orbit_on_set_with_given_action_after_restriction_by_length(
      actions::action *A_given, long int *Set, int set_sz,
      int desired_orbit_length,
      long int *&extracted_set,
      int verbose_level);
void extract_specific_orbit_on_set_with_given_action_after_restriction_by_lengt
h(
      actions::action *A_given, long int *Set, int set_sz,
      int desired_orbit_length,
      int desired_orbit_idx,
      long int *&extracted_set,
      int verbose_level);
void orbits_on_points_with_given_action(actions::action *A_given,
```

```
   int &nb_orbits, int *&orbit_reps, int verbose_level);
schreier *orbits_on_points_schreier(actions::action *A_given,
   int verbose_level);
schreier *orbit_of_one_point_schreier(actions::action *A_given,
   int pt, int verbose_level);
void orbits_light(actions::action *A_given,
   int *&Orbit_reps, int *&Orbit_lengths, int &nb_orbits,
   int **&Pts_per_generator, int *&Nb_per_generator,
   int verbose_level);
void write_to_file_binary(
      std::ofstream &fp, int verbose_level);
void read_from_file_binary(
      actions::action *A, std::ifstream &fp,
   int verbose_level);
void write_file(std::string &fname, int verbose_level);
void read_file(
      actions::action *A,
      std::string &fname, int verbose_level);
void compute_ascii_coding(
      std::string &ascii_coding, int verbose_level);
void decode_ascii_coding(
      std::string &ascii_coding, int verbose_level);
void export_permutation_group_to_magma(std::string &fname,
      actions::action *A2, int verbose_level);
void export_permutation_group_to_GAP(std::string &fname,
      actions::action *A2, int verbose_level);
void compute_and_print_orbits_on_a_given_set(
      actions::action *A_given,
   long int *set, int len, int verbose_level);
void compute_and_print_orbits(actions::action *A_given,
   int verbose_level);
int test_if_normalizing(sims *S, int verbose_level);
int test_if_subgroup(sims *S, int verbose_level);
void test_if_set_is_invariant_under_given_action(
      actions::action *A_given,
   long int *set, int set_sz, int verbose_level);
void set_of_coset_representatives(sims *S,
      data_structures_groups::vector_ge *&coset_reps,
      int verbose_level);
strong_generators *point_stabilizer(
      int pt, int verbose_level);
strong_generators *find_cyclic_subgroup_with_exactly_n_fixpoints(
      int nb_fixpoints,
      actions::action *A_given, int verbose_level);
void make_element_which_moves_a_point_from_A_to_B(
      actions::action *A_given,
   int pt_A, int pt_B, int *Elt, int verbose_level);
```

```
void export_group_to_GAP_and_copy_to_latex(
      std::string &label_txt,
      std::ostream &ost,
      actions::action *A2,
      int verbose_level);
void export_group_and_copy_to_latex(
      std::string &label_txt,
      std::ostream &ost,
      actions::action *A2,
      int verbose_level);
void report_fixed_objects_in_P3(
      std::ostream &ost,
      geometry::projective_space *P3,
      int verbose_level);
void reverse_isomorphism_exterior_square(int verbose_level);
void get_gens_data(int *&data, int &sz, int verbose_level);
void get_gens_data_as_string_with_quotes(
      std::string &str, int verbose_level);
void export_to_orbiter_as_bsgs(
      actions::action *A2,
      std::string &fname, std::string &label, std::string &label_tex,
      int verbose_level);
void report_group(std::string &prefix, int verbose_level);
void report_group2(std::ostream &ost, int verbose_level);


// strong_generators_groups.cpp
void prepare_from_generator_data(
      actions::action *A,
      int *data,
      int nb_gens,
      int data_size,
      std::string &ascii_target_go,
      int verbose_level);
void init_linear_group_from_scratch(
      actions::action *&A,
      field_theory::finite_field *F, int n,
   linear_group_description *Descr,
   data_structures_groups::vector_ge *&nice_gens,
   std::string &label,
   std::string &label_tex,
   int verbose_level);
void special_subgroup(int verbose_level);
void projectivity_subgroup(sims *S, int verbose_level);
void even_subgroup(int verbose_level);
void Sylow_subgroup(sims *S, int p, int verbose_level);
void init_single(actions::action *A, int *Elt, int verbose_level);
```

```
void init_single_with_target_go(actions::action *A,
      int *Elt, int target_go, int verbose_level);
void init_trivial_group(actions::action *A, int verbose_level);
void generators_for_the_monomial_group(actions::action *A,
   matrix_group *Mtx, int verbose_level);
void generators_for_the_diagonal_group(actions::action *A,
   matrix_group *Mtx, int verbose_level);
void generators_for_the_singer_cycle(actions::action *A,
   matrix_group *Mtx, int power_of_singer,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);
void generators_for_the_singer_cycle_and_the_Frobenius(
      actions::action *A,
   matrix_group *Mtx, int power_of_singer,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);
void generators_for_the_null_polarity_group(actions::action *A,
   matrix_group *Mtx, int verbose_level);
void generators_for_symplectic_group(actions::action *A,
   matrix_group *Mtx, int verbose_level);
void init_centralizer_of_matrix(actions::action *A, int *Mtx,
   int verbose_level);
void init_centralizer_of_matrix_general_linear(
      actions::action *A_projective,
      actions::action *A_general_linear, int *Mtx,
   int verbose_level);
void field_reduction(actions::action *Aq, int n, int s,
      field_theory::finite_field *Fq, int verbose_level);
void generators_for_translation_plane_in_andre_model(
      actions::action *A_PGL_n1_q, actions::action *A_PGL_n_q,
   matrix_group *Mtx_n1, matrix_group *Mtx_n,
   strong_generators *spread_stab_gens,
   int verbose_level);
void generators_for_the_stabilizer_of_two_components(
      actions::action *A_PGL_n_q,
   matrix_group *Mtx, int verbose_level);
void regulus_stabilizer(actions::action *A_PGL_n_q,
   matrix_group *Mtx, int verbose_level);
void generators_for_the_borel_subgroup_upper(
      actions::action *A_linear,
   matrix_group *Mtx, int verbose_level);
void generators_for_the_borel_subgroup_lower(
      actions::action *A_linear,
   matrix_group *Mtx, int verbose_level);
void generators_for_the_identity_subgroup(
      actions::action *A_linear,
   matrix_group *Mtx, int verbose_level);
```

```
void generators_for_parabolic_subgroup(
      actions::action *A_PGL_n_q,
   matrix_group *Mtx, int k, int verbose_level);
void generators_for_stabilizer_of_three_collinear_points_in_PGL4(
      actions::action *A_PGL_4_q,
   matrix_group *Mtx, int verbose_level);
void generators_for_stabilizer_of_triangle_in_PGL4(
      actions::action *A_PGL_4_q,
   matrix_group *Mtx, int verbose_level);
void generators_for_the_orthogonal_group(actions::action *A,
      field_theory::finite_field *F, int n,
   int epsilon,
   int f_semilinear,
   int verbose_level);
void stabilizer_of_cubic_surface_from_catalogue(
      actions::action *A,
   field_theory::finite_field *F, int iso,
   int verbose_level);
void init_reduced_generating_set(
      data_structures_groups::vector_ge *gens,
      ring_theory::longinteger_object &target_go,
      int verbose_level);
void stabilizer_of_quartic_curve_from_catalogue(
      actions::action *A,
   field_theory::finite_field *F, int iso,
   int verbose_level);
void stabilizer_of_Eckardt_surface(
      actions::action *A,
   field_theory::finite_field *F,
   int f_with_normalizer, int f_semilinear,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);
void stabilizer_of_G13_surface(
      actions::action *A,
   field_theory::finite_field *F, int a,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);
void stabilizer_of_F13_surface(
      actions::action *A,
   field_theory::finite_field *F, int a,
   data_structures_groups::vector_ge *&nice_gens,
   int verbose_level);
void BLT_set_from_catalogue_stabilizer(actions::action *A,
      field_theory::finite_field *F, int iso,
   int verbose_level);
void stabilizer_of_spread_from_catalogue(actions::action *A,
   int q, int k, int iso,
```

```
      int verbose_level);
   void stabilizer_of_pencil_of_conics(
         actions::action *A,
      field_theory::finite_field *F,
      int verbose_level);
   void Janko1(
         actions::action *A,
      field_theory::finite_field *F,
      int verbose_level);
   void Hall_reflection(
      int nb_pairs, int &degree, int verbose_level);
   void normalizer_of_a_Hall_reflection(
      int nb_pairs, int &degree, int verbose_level);
   void hyperplane_lifting_with_two_lines_fixed(
      strong_generators *SG_hyperplane,
      geometry::projective_space *P, int line1, int line2,
      int verbose_level);
   void exterior_square(
         actions::action *A_detached,
         strong_generators *SG_original,
         data_structures_groups::vector_ge *&nice_gens,
         int verbose_level);
   void diagonally_repeat(
         actions::action *An,
         strong_generators *Sn,
         int verbose_level);

};

// ##############################################################################
// subgroup.cpp:
// ##############################################################################

//! a subgroup of a group using a list of elements

class subgroup {
public:
   actions::action *A;
   int *Elements;
   long int group_order;
   int *gens;
   int nb_gens;
   sims *Sub;
   strong_generators *SG;


   subgroup();
```

```
   ~subgroup();
   void init_from_sims(sims *S, sims *Sub,
         strong_generators *SG, int verbose_level);
   void init(int *Elements, int group_order, int *gens, int nb_gens);
   void print();
   int contains_this_element(int elt);
   void report(std::ostream &ost);
};

// ##############################################################################
// sylow_structure.cpp:
// ##############################################################################

//! The Sylow structure of a finite group

class sylow_structure {
public:
   ring_theory::longinteger_object go;
   int *primes;
   int *exponents;
   int nb_primes;

   sims *S; // the group
   subgroup *Sub; // [nb_primes]

   sylow_structure();
   ~sylow_structure();
   void init(sims *S, int verbose_level);
   void report(std::ostream &ost);
};


// ##############################################################################
// wreath_product.cpp
// ##############################################################################

//! the wreath product group GL(d,q) wreath Sym(n)

class wreath_product {

public:
   matrix_group *M;
   actions::action *A_mtx;
   field_theory::finite_field *F;
   int q;
   int nb_factors;
```

```
std::string label;
std::string label_tex;

int degree_of_matrix_group;
   // = M->degree;
int dimension_of_matrix_group;
   // = M->n
int dimension_of_tensor_action;
   // = i_power_j(dimension_of_matrix_group, nb_factors);
long int degree_of_tensor_action;
   // = (i_power_j_safe(q, dimension_of_tensor_action) - 1) / (q - 1);
long int degree_overall;
   // = nb_factors + nb_factors *
   // degree_of_matrix_group + degree_of_tensor_action;
int low_level_point_size;
   // = dimension_of_tensor_action
int make_element_size;
   // = nb_factors + nb_factors *
   // dimension_of_matrix_group * dimension_of_matrix_group;
permutation_representation_domain *P;
int elt_size_int;
   // = M->elt_size_int * nb_factors + P->elt_size_int;

int *perm_offset_i; // [nb_factors + 1]
   // perm_offset_i[0] = nb_factors
   // perm_offset_i[nb_factors] = beginning of tensor domain
int *mtx_size;
int *index_set1;
int *index_set2;
int *u; // [dimension_of_tensor_action]
int *v; // [dimension_of_tensor_action]
int *w; // [dimension_of_tensor_action]
int *A1; // [dimension_of_tensor_action * dimension_of_tensor_action]
int *A2; // [dimension_of_tensor_action * dimension_of_tensor_action]
int *A3; // [dimension_of_tensor_action * dimension_of_tensor_action]
int *tmp_Elt1; // [elt_size_int]
int *tmp_perm1; // [P->elt_size_int]
int *tmp_perm2; // [P->elt_size_int]
int *induced_perm; // [dimension_of_tensor_action]

int bits_per_digit;
int bits_per_elt;
int char_per_elt;

uchar *elt1;

int base_len_in_component;
```

```
long int *base_for_component;
int *tl_for_component;

int base_length;
long int *the_base;
int *the_transversal_length;

data_structures::page_storage *Elts;

uint32_t *rank_one_tensors; // [nb_rank_one_tensors]
long int *rank_one_tensors_in_PG; // [nb_rank_one_tensors]
   // rank_one_tensors_in_PG[i] = affine_rank_to_PG_rank(rank_one_tensors[i]);
long int *rank_one_tensors_in_PG_sorted; // [nb_rank_one_tensors]
int nb_rank_one_tensors;

char *TR; // [degree_of_tensor_action + 1]
uint32_t *Prev; // [degree_of_tensor_action + 1]

wreath_product();
~wreath_product();
void init_tensor_wreath_product(matrix_group *M,
      actions::action *A_mtx, int nb_factors,
      int verbose_level);
void compute_tensor_ranks(int verbose_level);
void unrank_point(long int a, int *v, int verbose_level);
long int rank_point(int *v, int verbose_level);
long int element_image_of(int *Elt, long int a, int verbose_level);
void element_image_of_low_level(int *Elt,
      int *input, int *output, int verbose_level);
   // we assume that we are in the tensor product domain
void element_one(int *Elt);
int element_is_one(int *Elt);
void element_mult(int *A, int *B, int *AB, int verbose_level);
void element_move(int *A, int *B, int verbose_level);
void element_invert(int *A, int *Av, int verbose_level);
void compute_induced_permutation(int *Elt, int *perm);
void apply_permutation(int *Elt,
      int *v_in, int *v_out, int verbose_level);
int offset_i(int i);
void create_matrix(int *Elt, int *A, int verbose_level);
   // uses A1, A2
void element_pack(int *Elt, uchar *elt);
void element_unpack(uchar *elt, int *Elt);
void put_digit(uchar *elt, int f, int i, int j, int d);
int get_digit(uchar *elt, int f, int i, int j);
void make_element_from_one_component(int *Elt,
      int f, int *Elt_component);
```

```
void make_element_from_permutation(int *Elt, int *perm);
void make_element(int *Elt, int *data, int verbose_level);
void element_print_for_make_element(int *Elt, std::ostream &ost);
void element_print_easy(int *Elt, std::ostream &ost);
void element_print_latex(int *Elt, std::ostream &ost);
void compute_base_and_transversals(int verbose_level);
void make_strong_generators_data(int *&data,
      int &size, int &nb_gens, int verbose_level);
void report_rank_one_tensors(
      std::ostream &ost, int verbose_level);
void create_all_rank_one_tensors(
      uint32_t *&rank_one_tensors,
      int &nb_rank_one_tensors, int verbose_level);
uint32_t tensor_affine_rank(int *tensor);
void tensor_affine_unrank(int *tensor, uint32_t rk);
long int tensor_PG_rank(int *tensor);
void tensor_PG_unrank(int *tensor, long int PG_rk);
long int affine_rank_to_PG_rank(uint32_t affine_rk);
uint32_t PG_rank_to_affine_rank(long int PG_rk);
void save_rank_one_tensors(int verbose_level);
void compute_tensor_ranks(char *&TR, uint32_t *&Prev, int verbose_level);
void report(std::ostream &ost, int verbose_level);
void compute_permutations_and_write_to_file(
      strong_generators* SG,
      actions::action* A,
      int*& result,
      int &nb_gens, int &degree,
      int nb_factors,
      int verbose_level);
void make_fname(std::string &fname, int nb_factors, int h, int b);
int test_if_file_exists(int nb_factors, int h, int b);
void orbits_using_files_and_union_find(
      strong_generators* SG,
      actions::action* A,
      int *&result,
      int &nb_gens, int &degree,
      int nb_factors,
      int verbosity);
void orbits_restricted(
      strong_generators* SG,
      actions::action* A,
      int *&result,
      int &nb_gens, int &degree,
      int nb_factors,
      std::string &orbits_restricted_fname,
      int verbose_level);
void orbits_restricted_compute(
```

```
        strong_generators* SG,
        actions::action* A,
        int *&result,
        int &nb_gens, int &degree,
        int nb_factors,
        std::string &orbits_restricted_fname,
        int verbose_level);
};

}}}


#endif /* ORBITER_SRC_LIB_GROUP_ACTIONS_GROUPS_GROUPS_H_ */
```

5.5 Induced Actions

```
// induced_actions.h
//
// Anton Betten
//
// moved here from action.h: July 28, 2018
// based on action.h which was started:  August 13, 2005



#ifndef ORBITER_SRC_LIB_GROUP_ACTIONS_INDUCED_ACTIONS_INDUCED_ACTIONS_H_
#define ORBITER_SRC_LIB_GROUP_ACTIONS_INDUCED_ACTIONS_INDUCED_ACTIONS_H_



namespace orbiter {
namespace layer3_group_actions {
namespace induced_actions {


// ##############################################################################
// action_by_conjugation.cpp
// ##############################################################################


//! induced action by conjugation on the elements of a given group



class action_by_conjugation {
public:
   groups::sims *Base_group;
   int f_ownership;
   long int goi;

   int *Elt1;
   int *Elt2;
   int *Elt3;

   action_by_conjugation();
   ~action_by_conjugation();
   void init(
         groups::sims *Base_group,
         int f_ownership, int verbose_level);
   long int compute_image(
         actions::action *A,
         int *Elt, long int i, int verbose_level);
```

```
   long int rank(int *Elt);
   long int multiply(
         actions::action *A,
         long int i, long int j, int verbose_level);
};

// ##############################################################################
// action_by_representation.cpp
// ##############################################################################


//! induced  action of PSL(2,q) on a conic (the only type implemented so far)


class action_by_representation {
public:
   enum representation_type type;
   int n;
   int q;
   groups::matrix_group *M;
   field_theory::finite_field *F;
   int low_level_point_size;
   int degree;

   int dimension; //
   int *v1; // [dimension]
   int *v2; // [dimension]
   int *v3; // [dimension]

   action_by_representation();
   ~action_by_representation();
   void init_action_on_conic(
         actions::action &A, int verbose_level);
   long int compute_image_int(
         actions::action &A, int *Elt,
         long int a, int verbose_level);
   void compute_image_int_low_level(
         actions::action &A, int *Elt,
         int *input, int *output,
      int verbose_level);
   void unrank_point(long int a, int *v, int verbose_level);
   long int rank_point(int *v, int verbose_level);

};

// ##############################################################################
// action_by_restriction.cpp
```

```
// ##############################################################################


//! restricted action on an invariant subset




class action_by_restriction {
public:
   int nb_points;
   long int *points; // [nb_points]
   long int *points_sorted; // [nb_points]
   long int *perm_inv; // [nb_points]
   int f_single_orbit;
   int pt;
   int idx_of_root_node;

   action_by_restriction();
   ~action_by_restriction();
   void init_single_orbit_from_schreier_vector(
         data_structures_groups::schreier_vector *Schreier_vector,
         int pt, int verbose_level);
   void init(
         int nb_points,
         long int *points, int verbose_level);
      // the array points must be ordered
   long int original_point(long int pt);
   long int restricted_point_idx(long int pt);
   long int compute_image(actions::action *A,
         int *Elt, long int i, int verbose_level);
};

// ##############################################################################
// action_by_right_multiplication.cpp
// ##############################################################################

//! induced action on a the set of elements of a group by right multiplication



class action_by_right_multiplication {
public:
   groups::sims *Base_group;
   int f_ownership;
   int goi;
```

```
   int *Elt1;
   int *Elt2;

   action_by_right_multiplication();
   ~action_by_right_multiplication();
   void init(
         groups::sims *Base_group,
         int f_ownership, int verbose_level);
   long int compute_image(
         actions::action *A, int *Elt,
         long int i,
      int verbose_level);
};

// ##############################################################################
// action_by_subfield_structure.cpp
// ##############################################################################

//! induced action on the vector space arising from a field over a subfield


class action_by_subfield_structure {
public:
   int n;
   int Q;
   const char *poly_q;
   int q;
   int s;
   int m; // n * s
   int *v1; // [m]
   int *v2; // [m]
   int *v3; // [m]

   actions::action *AQ;
   actions::action *Aq;

   groups::matrix_group *MQ;
   field_theory::finite_field *FQ;
   groups::matrix_group *Mq;
   field_theory::finite_field *Fq;

   field_theory::subfield_structure *S;

   int *Eltq;
   int *Mtx; // [m * m]

   int low_level_point_size; // = m
```

```
   int degree;

   action_by_subfield_structure();
   ~action_by_subfield_structure();
   void init(
         actions::action &A,
         field_theory::finite_field *Fq,
         int verbose_level);
   long int compute_image_int(
         actions::action &A, int *Elt,
         long int a, int verbose_level);
   void compute_image_int_low_level(
         actions::action &A, int *Elt,
         int *input, int *output,
      int verbose_level);
};

// ##############################################################################
// action_on_andre.cpp
// ##############################################################################

//! induced action on the elements of a projective plane constructed via Andre /
Bruck / Bose


class action_on_andre {
public:

   actions::action *An;
   actions::action *An1;
   geometry::andre_construction *Andre;
   int k, n, q;
   int k1, n1;
   int N; // number of points in the plane
   int degree;
   int *coords1; // [(k + 1) * (n + 1)];
   int *coords2; // [(k + 1) * (n + 1)];
   int *coords3; // [k * n];

   action_on_andre();
   ~action_on_andre();
   void init(
         actions::action *An,
         actions::action *An1,
         geometry::andre_construction *Andre,
         int verbose_level);
   long int compute_image(
```

```
         int *Elt, long int i,
      int verbose_level);
   long int compute_image_of_point(
         int *Elt, long int pt_idx,
      int verbose_level);
   long int compute_image_of_line(
         int *Elt, long int line_idx,
      int verbose_level);
};

// ##############################################################################
// action_on_bricks.cpp
// ##############################################################################

//! related to a problem of Neil Sloane


class action_on_bricks {
public:

   actions::action *A;
   combinatorics::brick_domain *B;
   int degree;
   int f_linear_action;

   action_on_bricks();
   ~action_on_bricks();
   void init(
         actions::action *A,
         combinatorics::brick_domain *B,
         int f_linear_action,
      int verbose_level);
   long int compute_image(int *Elt, long int i,
      int verbose_level);
   long int compute_image_linear_action(
         int *Elt, long int i,
      int verbose_level);
   long int compute_image_permutation_action(
         int *Elt, long int i,
      int verbose_level);
};

// ##############################################################################
// action_on_cosets.cpp
// ##############################################################################

//! induced action on the cosets of a subspace by right multiplication
```

```
class action_on_cosets {
public:
   actions::action *A_linear;
   field_theory::finite_field *F;
   int dimension_of_subspace;
   int n;
   int *subspace_basis; // [dimension_of_subspace * n]
   int *base_cols; // [dimension_of_subspace]
      // the pivot column for the subspace basis
      // to be used if a vector v[len]
      // is reduced modulo a subspace

   int f_lint;
   int nb_points;
   int *Points; // ordered list of point ranks
   long int *lint_Points; // ordered list of point ranks

   int *v1;
   int *v2;

   void (*unrank_point)(int *v, int a, void *data);
   int (*rank_point)(int *v, void *data);
   void (*unrank_point_lint)(int *v, long int a, void *data);
   long int (*rank_point_lint)(int *v, void *data);
   void *rank_unrank_data;

   action_on_cosets();
   ~action_on_cosets();
   void init(int nb_points, int *Points,
         actions::action *A_linear,
      field_theory::finite_field *F,
      int dimension_of_subspace,
      int n,
      int *subspace_basis,
      int *base_cols,
      void (*unrank_point)(int *v, int a, void *data),
      int (*rank_point)(int *v, void *data),
      void *rank_unrank_data,
      int verbose_level);
   void init_lint(int nb_points, long int *Points,
         actions::action *A_linear,
      field_theory::finite_field *F,
      int dimension_of_subspace,
      int n,
      int *subspace_basis,
```

```
      int *base_cols,
      void (*unrank_point)(int *v, long int a, void *data),
      long int (*rank_point)(int *v, void *data),
      void *rank_unrank_data,
      int verbose_level);
   void reduce_mod_subspace(int *v, int verbose_level);
   long int compute_image(
         int *Elt, long int i, int verbose_level);

};

// ##############################################################################
// action_on_determinant.cpp
// ##############################################################################

//! induced action on the determinant of a group of matrices (used to compute the
 subgroup PSL)


class action_on_determinant {
public:
   groups::matrix_group *M;
   int f_projective;
   int m;
   int q;
   int degree;
      // gcd(m, q - 1) if f_projective
      // q - 1 otherwise

   action_on_determinant();
   ~action_on_determinant();
   void init(
         actions::action &A,
         int f_projective, int m, int verbose_level);
   long int compute_image(
         actions::action *A, int *Elt, long int i,
      int verbose_level);
};

// ##############################################################################
// action_on_factor_space.cpp
// ##############################################################################

//! induced action on the factor space of a vector space modulo a subspace


class action_on_factor_space {
```

```
public:
   algebra::vector_space *VS;


   // VS->dimension = length of vectors in large space

   int *subspace_basis; // [subspace_basis_size * VS->dimension]
   int subspace_basis_size;
   int *base_cols; // [subspace_basis_size]
      // the pivot column for the subspace basis
      // to be used if a vector v[len] is reduced modulo a subspace

   long int degree;
      // the number of projective points in the small space
      // (i.e., the factor space)
      // (q^factor_space_len - 1) / (q - 1),
      // as computed by compute_degree();
   long int large_degree;
      // the number of projective points in the large space
      // (q^len - 1) / (q - 1),
      // as computed by compute_large_degree();

   int factor_space_len;
      // = VS->dimension - subspace_basis_size

   int *embedding;
      // [factor_space_len]
      // the list of columns that are not pivot columns,
      // i.e. not in base_cols[]
      // this is the set-theoretic complement of base_cols
   long int *projection_table;
      // [nb_points]
      // projection_table[i] = j
      // means that the Gauss reduced vector in
      // the coset of point_list[i]
      // is in coset_reps_Gauss[j]
   long int *preimage_table; // [degree]
   int *tmp; // [factor_space_len]
   int *Tmp1; // [VS->dimension]
   int *Tmp2; // [VS->dimension]
   int f_tables_have_been_computed;

   int f_table_mode;

   int nb_cosets;
   long int *coset_reps_Gauss;
      // [nb_cosets]
```

```
   // ordered list of Gauss reduced coset representatives
   // the entries are ranks of vectors in the large space

int *tmp_w; // [VS->dimension] temporary vector for use in rank
int *tmp_w1;
   // [subspace_basis_size]
   // temporary vector for lexleast_element_in_coset
int *tmp_v1;
   // [len] temporary vector
   // for use in lexleast_element_in_coset
int *tmp_v2;
   // [len] temporary vector
   // for use in lexleast_element_in_coset

action_on_factor_space();
~action_on_factor_space();
void init_light(
   algebra::vector_space *VS,
   actions::action &A_base, actions::action &A,
   long int *subspace_basis_ranks, int subspace_basis_size,
   int verbose_level);
void init_by_rank_table_mode(
      algebra::vector_space *VS,
      actions::action &A_base, actions::action &A,
   long int *subspace_basis_ranks, int subspace_basis_size,
   long int *point_list, int nb_points,
   int verbose_level);
void print_coset_table();
void print_projection_table(
      long int *point_list, int nb_points);
void init_coset_table(
      long int *point_list, int nb_points,
      int verbose_level);
void init_by_rank(
      algebra::vector_space *VS,
      actions::action &A_base, actions::action &A,
   long int *subspace_basis_ranks, int subspace_basis_size,
   int f_compute_tables, int verbose_level);
void init_from_coordinate_vectors(
      algebra::vector_space *VS,
      actions::action &A_base, actions::action &A,
   int *subspace_basis, int subspace_basis_size,
   int f_compute_tables, int verbose_level);
void init2(actions::action &A_base, actions::action &A,
   int f_compute_tables, int verbose_level);
void compute_projection_table(int verbose_level);
long int compute_degree();
```

```
   long int compute_large_degree();
   void list_all_elements();
   void reduce_mod_subspace(int *v, int verbose_level);
   long int lexleast_element_in_coset(
         long int rk, int verbose_level);
      // This function computes the lexleast
      // element in the coset modulo the subspace.
      // It does so by looping over all q^subspace_basis_size
      // elements in the subspace and ranking the corresponding
      // vector in the large space using rank_in_large_space(v2).
   long int project_onto_Gauss_reduced_vector(
         long int rk, int verbose_level);
   long int project(long int rk, int verbose_level);
      // unranks the vector rk, and reduces it
      // modulo the subspace basis.
      // The non-pivot components are considered
      // as a vector in F_q^factor_space_len
      // and ranked using the rank function for projective space.
      // This rank is returned.
      // If the vector turns out to lie in the
      // subspace, -1 is returned.
   long int preimage(long int rk, int verbose_level);
   void embed(int *from, int *to);
   void unrank(int *v, long int rk, int verbose_level);
   long int rank(int *v, int verbose_level);
   void unrank_in_large_space(int *v, long int rk);
   long int rank_in_large_space(int *v);
   void unrank_in_small_space(int *v, long int rk);
   long int rank_in_small_space(int *v);
   long int compute_image(
         actions::action *A,
         int *Elt, long int i, int verbose_level);
};

// ##############################################################################
// action_on_flags.cpp
// ##############################################################################


//! induced action on flags


class action_on_flags {
public:
   actions::action *A;
   int n;
   field_theory::finite_field *F;
```

```
   int *type;
   int type_len;
   geometry::flag *Flag;
   groups::matrix_group *M;
   int degree;
   int *M1;
   int *M2;

   action_on_flags();
   ~action_on_flags();
   void init(
         actions::action *A,
         int *type, int type_len,
      int verbose_level);
   long int compute_image(
         int *Elt,
         long int i, int verbose_level);
};

// ##############################################################################
// action_on_galois_group.cpp:
// ##############################################################################

//! induced action on the galois group (used to compute the projectivity subgroup
 of a collineation group)


class action_on_galois_group {
public:
   actions::action *A;
   groups::matrix_group *M;
   int m;
   int q;
   int degree;

   action_on_galois_group();
   ~action_on_galois_group();
   void init(
         actions::action *A,
         int m, int verbose_level);
   long int compute_image(int *Elt, long int i,
      int verbose_level);
};


// ##############################################################################
// action_on_grassmannian.cpp
```

```
// ##############################################################################

//! induced action on the grassmannian (subspaces of a fixed dimension of a vecto
r space)


class action_on_grassmannian {
public:
   int n;
   int k;
   int q;
   field_theory::finite_field *F;
   int low_level_point_size;

   actions::action *A;
   geometry::grassmann *G;
   int *M1;
   int *M2;

   int f_embedding;
   int big_n;
   geometry::grassmann_embedded *GE;
   int *subspace_basis; // [n * big_n]
   int *subspace_basis2; // [n * big_n]

   ring_theory::longinteger_object degree;
   int max_string_length;

   int f_has_print_function;
   void (*print_function)(
         std::ostream &ost, long int a, void *data);
   void *print_function_data;

   action_on_grassmannian();
   ~action_on_grassmannian();
   void init(actions::action &A,
         geometry::grassmann *G, int verbose_level);
   void add_print_function(
         void (*print_function)(
               std::ostream &ost, long int a, void *data),
         void *print_function_data,
         int verbose_level);
   void init_embedding(int big_n, int *ambient_space,
      int verbose_level);
   void unrank(long int i, int *v, int verbose_level);
   long int rank(int *v, int verbose_level);
   void compute_image_longinteger(
```

```
        actions::action *A,
        int *Elt,
        ring_theory::longinteger_object &i,
        ring_theory::longinteger_object &j,
      int verbose_level);
   long int compute_image_int(
        actions::action *A, int *Elt,
      long int i, int verbose_level);
   long int compute_image_int_ordinary(
        actions::action *A, int *Elt,
      long int i, int verbose_level);
   long int compute_image_int_embedded(
        actions::action *A, int *Elt,
      long int i, int verbose_level);
   void print_point(long int a, std::ostream &ost);
};

// ##############################################################################
// action_on_homogeneous_polynomials.cpp
// ##############################################################################


//! induced action on the set of homogeneous polynomials over a finite field


class action_on_homogeneous_polynomials {
public:
   int n; // the dimension M->n
   int q;
   actions::action *A;
   ring_theory::homogeneous_polynomial_domain *HPD;
   groups::matrix_group *M;
   field_theory::finite_field *F;
   int low_level_point_size;
   int degree;

   // wedge product
   int dimension; // = HPD->nb_monomials
   int *v1; // [dimension]
   int *v2; // [dimension]
   int *v3; // [dimension]
   int *Elt1;

   int f_invariant_set;
   int *Equations;
   int nb_equations;
```

```
   data_structures::int_matrix *Table_of_equations;

   action_on_homogeneous_polynomials();
   ~action_on_homogeneous_polynomials();
   void init(
         actions::action *A,
         ring_theory::homogeneous_polynomial_domain *HPD,
      int verbose_level);
   void init_invariant_set_of_equations(
         int *Equations,
      int nb_equations, int verbose_level);
   void unrank_point(int *v, long int rk);
   long int rank_point(int *v);
   long int compute_image_int(
         int *Elt, long int a, int verbose_level);
   void compute_image_int_low_level(
      int *Elt, int *input, int *output, int verbose_level);
   void compute_representation(
      int *Elt, int *M, int verbose_level);
};

// ##############################################################################
// action_on_interior_direct_product.cpp
// ##############################################################################


//! induced action on the interior direct product


class action_on_interior_direct_product {
public:
   actions::action *A;
   int nb_rows;
   int nb_cols;
   int degree;


   action_on_interior_direct_product();
   ~action_on_interior_direct_product();
   void init(
         actions::action *A,
         int nb_rows, int verbose_level);
   long int compute_image(
         int *Elt, long int a, int verbose_level);
};
```

```
// ##############################################################################
// action_on_k_subsets.cpp
// ##############################################################################

//! induced action on k-subsets of a set of size n


class action_on_k_subsets {
public:
   actions::action *A;
   int k;
   int degree;
   int *set1; // [k]
   int *set2; // [k]

   action_on_k_subsets();
   ~action_on_k_subsets();
   void init(
         actions::action *A, int k, int verbose_level);
   long int compute_image(
         int *Elt, long int i, int verbose_level);
};

// ##############################################################################
// action_on_orbits.cpp
// ##############################################################################

//! induced action on the set of orbits (usually by the normalizer)



class action_on_orbits {
public:
   actions::action *A;
   groups::schreier *Sch;
   int f_play_it_safe;
   int degree;

   action_on_orbits();
   ~action_on_orbits();
   void init(
         actions::action *A,
         groups::schreier *Sch,
         int f_play_it_safe,
      int verbose_level);
   long int compute_image(
```

```
         int *Elt, long int i, int verbose_level);
};

// ##############################################################################
// action_on_orthogonal.cpp
// ##############################################################################


//! induced action on the orthogonal geometry


class action_on_orthogonal {
public:
   actions::action *original_action;
   orthogonal_geometry::orthogonal *O;
   int *v1;
   int *v2;
   int *w1;
   int *w2;
   int f_on_points;
   int f_on_lines;
   int f_on_points_and_lines;
   int low_level_point_size;
   int degree;

   action_on_orthogonal();
   ~action_on_orthogonal();
   void init(
         actions::action *original_action,
         orthogonal_geometry::orthogonal *O,
      int f_on_points, int f_on_lines,
      int f_on_points_and_lines,
      int verbose_level);
   void unrank_point(int *v, int rk);
   int rank_point(int *v);
   long int map_a_point(
         int *Elt, long int i, int verbose_level);
   long int map_a_line(
         int *Elt, long int i, int verbose_level);
   long int compute_image_int(
         int *Elt, long int i, int verbose_level);
};

// ##############################################################################
// action_on_set_partitions.cpp:
// ##############################################################################
```

```
//! induced action on a set partitions.



class action_on_set_partitions {
public:
   int nb_set_partitions;
   int universal_set_size;
   int partition_class_size;
   int nb_parts;
   actions::action *A;
   int *v1;
   int *v2;

   action_on_set_partitions();
   ~action_on_set_partitions();
   void init(int partition_size,
         actions::action *A,
      int verbose_level);
   long int compute_image(
      int *Elt,
      long int a, int verbose_level);
};

// ##############################################################################
// action_on_sets.cpp
// ##############################################################################


//! induced action on a given set of sets.



class action_on_sets {
public:
   int nb_sets;
   int set_size;
   long int **sets;
   long int *image_set;
   int *perm;
   int *perm_inv;

   action_on_sets();
   ~action_on_sets();
   void init(int nb_sets, int set_size,
      long int *input_sets, int verbose_level);
```

```
   int find_set(long int *set, int verbose_level);
   long int compute_image(
         actions::action *A, int *Elt,
      long int i, int verbose_level);
   void print_sets_sorted();
   void print_sets_in_original_ordering();
   void test_sets();
};

int action_on_sets_compare(void *a, void *b, void *data);
int action_on_sets_compare_inverted(void *a, void *b, void *data);

// ##############################################################################
// action_on_sign.cpp
// ##############################################################################

//! induced action on the sign function of a permutation group (to compute the ev
en subgroup)


class action_on_sign {
public:
   actions::action *A;
   int perm_degree;
   int *perm; // [perm_degree]
   int degree; // 2

   action_on_sign();
   ~action_on_sign();
   void init(
         actions::action *A, int verbose_level);
   long int compute_image(
         int *Elt, long int i, int verbose_level);
};

// ##############################################################################
// action_on_spread_set.cpp
// ##############################################################################


//! induced action on a spread set via the associated spread


class action_on_spread_set {
public:

   int k;
```

```
   int n; // = 2 * k
   int k2; // = k^2
   int q;
   field_theory::finite_field *F;
   int low_level_point_size; // = k * k
   int degree;

   actions::action *A_PGL_n_q;
   actions::action *A_PGL_k_q;
   groups::sims *G_PGL_k_q;

   int *Elt1;
   int *Elt2;

   int *mtx1; // [k * k]
   int *mtx2; // [k * k]
   int *subspace1; // [k * n]
   int *subspace2; // [k * n]

   action_on_spread_set();
   ~action_on_spread_set();
   void init(
         actions::action *A_PGL_n_q,
         actions::action *A_PGL_k_q,
         groups::sims *G_PGL_k_q,
      int k, field_theory::finite_field *F,
      int verbose_level);
   void report(
         std::ostream &ost, int verbose_level);
   long int compute_image_int(
         int *Elt, long int rk, int verbose_level);
   void matrix_to_subspace(int *mtx, int *subspace, int verbose_level);
   void subspace_to_matrix(int *subspace, int *mtx, int verbose_level);
   void unrank_point(long int rk, int *mtx, int verbose_level);
   long int rank_point(int *mtx, int verbose_level);
   void compute_image_low_level(
         int *Elt, int *input, int *output,
      int verbose_level);
};

// ##############################################################################
// action_on_subgroups.cpp
// ##############################################################################

//! induced action on subgroups of a group
```

```
class action_on_subgroups {
public:

   actions::action *A;
   groups::sims *S;
   int nb_subgroups;
   int subgroup_order;
   groups::subgroup **Subgroups;
   int **sets;
   int *image_set; // [subgroup_order]
   int *perm;
   int *perm_inv;
   int *Elt1;

   action_on_subgroups();
   ~action_on_subgroups();
   void init(
         actions::action *A,
         groups::sims *S, int nb_subgroups,
      int subgroup_order, groups::subgroup **Subgroups,
      int verbose_level);
   long int compute_image(
         int *Elt, long int a, int verbose_level);

};

int action_on_subgroups_compare(void *a, void *b, void *data);
int action_on_subgroups_compare_inverted(void *a, void *b, void *data);

// ##############################################################################
// action_on_wedge_product.cpp
// ##############################################################################


//! induced wedge product action on the exterior square of a vector space


class action_on_wedge_product {
public:

   actions::action *A;
   int n;
   int q;
   groups::matrix_group *M;
   field_theory::finite_field *F;
   int low_level_point_size;
   long int degree;
```

```
   // wedge product
   int wedge_dimension; // {n \choose 2}
   int *wedge_v1; // [wedge_dimension]
   int *wedge_v2; // [wedge_dimension]
   int *wedge_v3; // [wedge_dimension]
   int *Mtx_wedge; // [wedge_dimension * wedge_dimension]

   action_on_wedge_product();
   ~action_on_wedge_product();
   void init(actions::action *A, int verbose_level);
   void unrank_point(int *v, long int rk);
   long int rank_point(int *v);
   long int compute_image_int(
         int *Elt, long int a, int verbose_level);
   int element_entry_frobenius(int *Elt,
      int verbose_level);
   int element_entry_ij(int *Elt, int I, int J,
      int verbose_level);
   int element_entry_ijkl(int *Elt,
      int i, int j, int k, int l, int verbose_level);
   void compute_image_int_low_level(
         int *Elt, int *input, int *output,
      int verbose_level);
   void create_induced_matrix(
         int *Elt, int *Mtx2, int verbose_level);
   void element_print_latex(int *A, std::ostream &ost);
};


// ##############################################################################
// product_action.cpp
// ##############################################################################

//! induced product action of two group actions


class product_action {
public:
   actions::action *A1;
   actions::action *A2;
   int f_use_projections;
   int offset;
   int degree;
   int elt_size_in_int;
   int coded_elt_size_in_char;
```

```
   int *Elt1, *Elt2, *Elt3;
      // temporary storage
   uchar *elt1, *elt2, *elt3;
      // temporary storage, used in element_store()

   data_structures::page_storage *Elts;

   product_action();
   ~product_action();
   void init(
         actions::action *A1,
         actions::action *A2, int f_use_projections,
      int verbose_level);
   long int compute_image(
         actions::action *A,
         int *Elt, long int i, int verbose_level);
   void element_one(
         actions::action *A, int *Elt, int verbose_level);
   int element_is_one(
         actions::action *A, int *Elt, int verbose_level);
   void element_unpack(
         uchar *elt, int *Elt, int verbose_level);
   void element_pack(
         int *Elt, uchar *elt, int verbose_level);
   void element_retrieve(
         actions::action *A, int hdl, int *Elt,
      int verbose_level);
   int element_store(
         actions::action *A, int *Elt, int verbose_level);
   void element_mult(
         int *A, int *B, int *AB, int verbose_level);
   void element_invert(
         int *A, int *Av, int verbose_level);
   void element_transpose(
         int *A, int *At, int verbose_level);
   void element_move(
         int *A, int *B, int verbose_level);
   void element_print(
         int *A, std::ostream &ost);
   void element_print_latex(
         int *A, std::ostream &ost);
   void make_element(
         int *Elt, int *data, int verbose_level);
};


}}}
```

```
#endif /* ORBITER_SRC_LIB_GROUP_ACTIONS_INDUCED_ACTIONS_INDUCED_ACTIONS_H_ */
```

5.6 Interfaces

```
/*
 * interfaces.h
 *
 *  Created on: Jan 28, 2023
 *      Author: betten
 */

#ifndef SRC_LIB_LAYER3_GROUP_ACTIONS_INTERFACES_L3_INTERFACES_H_
#define SRC_LIB_LAYER3_GROUP_ACTIONS_INTERFACES_L3_INTERFACES_H_


namespace orbiter {

namespace layer3_group_actions {


namespace interfaces {


// ##############################################################################
// nauty_interface_with_group.cpp:
// ##############################################################################

//! Interface to GAP and fining at level 3


class l3_interface_gap {

public:

   l3_interface_gap();
   ~l3_interface_gap();
   void canonical_image_GAP(
         groups::strong_generators *SG,
         std::string &input_set_text,
         std::ostream &ost, int verbose_level);
   void export_collineation_group_to_fining(
         std::ostream &ost,
         groups::strong_generators *SG,
         int verbose_level);

};


// ##############################################################################
// magma_interface.cpp
```

```
// ##############################################################################


//! interface for group theoretic computations with the group theory software mag
ma

class magma_interface {

public:
   magma_interface();
   ~magma_interface();
   void centralizer_of_element(
         actions::action *A, groups::sims *S,
         std::string &element_description,
         std::string &label, int verbose_level);
   void normalizer_of_cyclic_subgroup(
         actions::action *A, groups::sims *S,
         std::string &element_description,
         std::string &label, int verbose_level);
   void find_subgroups(
         actions::action *A, groups::sims *S,
         int subgroup_order,
         std::string &label,
         int &nb_subgroups,
         groups::strong_generators *&H_gens,
         groups::strong_generators *&N_gens,
         int verbose_level);
   void print_generators_MAGMA(actions::action *A,
         groups::strong_generators *SG, std::ostream &ost);
   void export_group(actions::action *A,
         groups::strong_generators *SG, std::ostream &ost, int verbose_level);
   void export_permutation_group_to_magma(
         std::string &fname, actions::action *A2,
         groups::strong_generators *SG, int verbose_level);
   void export_permutation_group_to_magma2(
         std::ostream &ost,
         actions::action *A2,
         groups::strong_generators *SG,
         int verbose_level);
   void export_group_to_magma_and_copy_to_latex(
         std::string &label_txt,
         std::ostream &ost,
         actions::action *A2,
         groups::strong_generators *SG,
         int verbose_level);
   void normalizer_using_MAGMA(
         actions::action *A,
```

```
      std::string &fname_magma_prefix,
      groups::sims *G, groups::sims *H,
      groups::strong_generators *&gens_N,
      int verbose_level);
void conjugacy_classes_using_MAGMA(
      actions::action *A,
      std::string &prefix,
      groups::sims *G, int verbose_level);
void conjugacy_classes_and_normalizers_using_MAGMA_make_fnames(
      actions::action *A,
      std::string &prefix,
      std::string &fname_magma,
      std::string &fname_output);
void conjugacy_classes_and_normalizers_using_MAGMA(
      actions::action *A,
      std::string &prefix,
      groups::sims *G, int verbose_level);
void read_conjugacy_classes_and_normalizers_from_MAGMA(
      actions::action *A,
      std::string &fname,
      int &nb_classes,
      int *&perms,
      long int *&class_size,
      int *&class_order_of_element,
      long int *&class_normalizer_order,
      int *&class_normalizer_number_of_generators,
      int **&normalizer_generators_perms,
      int verbose_level);
void normalizer_of_cyclic_group_using_MAGMA(
      actions::action *A,
      std::string &fname_magma_prefix,
      groups::sims *G, int *Elt,
      groups::strong_generators *&gens_N,
      int verbose_level);
void centralizer_using_MAGMA(
      actions::action *A,
      std::string &prefix,
      groups::sims *override_Sims, int *Elt,
      groups::strong_generators *&gens,
      int verbose_level);
void read_centralizer_magma(
      actions::action *A,
      std::string &fname_output,
      groups::sims *override_Sims,
      groups::strong_generators *&gens,
      int verbose_level);
void centralizer_using_magma2(
```

```
        actions::action *A,
        std::string &prefix,
        std::string &fname_magma,
        std::string &fname_output,
        groups::sims *override_Sims, int *Elt,
        int verbose_level);
void find_subgroups_using_MAGMA(
        actions::action *A,
        std::string &prefix,
        groups::sims *override_Sims,
        int subgroup_order,
        int &nb_subgroups,
        groups::strong_generators *&H_gens,
        groups::strong_generators *&N_gens,
        int verbose_level);
void read_subgroups_magma(
        actions::action *A,
        std::string &fname_output,
        groups::sims *override_Sims, int subgroup_order,
        int &nb_subgroups,
        groups::strong_generators *&H_gens,
        groups::strong_generators *&N_gens,
        int verbose_level);
void find_subgroups_using_MAGMA2(
        actions::action *A,
        std::string &prefix,
        std::string &fname_magma, std::string &fname_output,
        groups::sims *override_Sims, int subgroup_order,
        int verbose_level);
void conjugacy_classes_and_normalizers(
        actions::action *A,
        groups::sims *override_Sims,
        std::string &label,
        std::string &label_tex,
        int verbose_level);
void report_conjugacy_classes_and_normalizers(
        actions::action *A,
        std::ostream &ost,
        groups::sims *override_Sims, int verbose_level);
void read_conjugacy_classes_and_normalizers(
        actions::action *A,
        std::string &fname, groups::sims *override_sims,
        std::string &label_latex, int verbose_level);
void read_and_report_conjugacy_classes_and_normalizers(
        actions::action *A,
        std::ostream &ost,
        std::string &fname, groups::sims *override_Sims,
```

```
         int verbose_level);
   void write_as_magma_permutation_group(groups::sims *S,
         std::string &fname_base,
         data_structures_groups::vector_ge *gens, int verbose_level);
   void export_linear_code(
         std::string &fname,
         field_theory::finite_field *F,
         int *genma, int n, int k,
         int verbose_level);
   void read_permutation_group(std::string &fname,
      int degree, int *&gens, int &nb_gens, int &go,
      int verbose_level);
   void run_magma_file(std::string &fname, int verbose_level);
   void normalizer_in_Sym_n(
         std::string &fname_base,
      int group_order, int *Table, int *gens, int nb_gens,
      int *&N_gens, int &N_nb_gens, int &N_go,
      int verbose_level);


};



// ##############################################################################
// nauty_interface_with_group.cpp:
// ##############################################################################

//! Interface to Nauty for computing canonical forms and automorphism groups of g
raphs


class nauty_interface_with_group {
public:
   nauty_interface_with_group();
   ~nauty_interface_with_group();
   actions::action *create_automorphism_group_of_colored_graph_object(
         graph_theory::colored_graph *CG, int verbose_level);
   actions::action *create_automorphism_group_and_canonical_labeling_of_colored_gr
aph_object(
         graph_theory::colored_graph *CG, int *labeling, int verbose_level);
   actions::action *create_automorphism_group_and_canonical_labeling_of_colored_gr
aph(
      int n, int f_bitvec, data_structures::bitvector *Bitvec, int *Adj,
      int *vertex_colors,
      int *labeling,
      int verbose_level);
```

```
   actions::action *create_automorphism_group_of_graph_bitvec(
      int n, data_structures::bitvector *Bitvec,
      int verbose_level);
   actions::action *create_automorphism_group_of_graph_with_partition_and_labeling
(
      int n,
      int f_bitvector, data_structures::bitvector *Bitvec, int *Adj,
      int nb_parts, int *parts,
      int *labeling,
      int verbose_level);
   actions::action *create_automorphism_group_of_graph(int *Adj,
      int n, int verbose_level);
   actions::action *create_automorphism_group_and_canonical_labeling_of_graph(
      int *Adj, int n, int *labeling, int verbose_level);
   // labeling[n]
   void automorphism_group_as_permutation_group(
         //strong_generators *&SG,
         data_structures::nauty_output *NO,
         actions::action *&A_perm,
         int verbose_level);
   void reverse_engineer_linear_group_from_permutation_group(
         actions::action *A_linear,
         geometry::projective_space *P,
         groups::strong_generators *&SG,
         actions::action *&A_perm,
         data_structures::nauty_output *NO,
         int verbose_level);
   groups::strong_generators *set_stabilizer_of_object(
         geometry::object_with_canonical_form *OwCF,
         actions::action *A_linear,
      int f_compute_canonical_form, data_structures::bitvector *&Canonical_form,
      data_structures::nauty_output *&NO,
      int verbose_level);

};


}}}



#endif /* SRC_LIB_LAYER3_GROUP_ACTIONS_INTERFACES_L3_INTERFACES_H_ */
```

Chapter 6

Layer 4 – Classification

6.1 Main Header File

```
// classification.h
//
// Anton Betten
//
// started:  September 20, 2007


#ifndef ORBITER_SRC_LIB_CLASSIFICATION_CLASSIFICATION_H_
#define ORBITER_SRC_LIB_CLASSIFICATION_CLASSIFICATION_H_



using namespace orbiter::layer1_foundations;
using namespace orbiter::layer2_discreta;
using namespace orbiter::layer3_group_actions;


namespace orbiter {

//! classification of combinatorial objects

namespace layer4_classification {


//! classification by using an invariant relation

namespace invariant_relations {

   // classify
   class classification_step;
   class flag_orbit_node;
```

```
   class flag_orbits;
   class orbit_node;

}


//! classification by using substructures and lifting

namespace isomorph {

   // isomorph
   class flag_orbit_folding;
   class isomorph_arguments;
   class isomorph;
   class isomorph_worker;
   struct isomorph_worker_data;
   class representatives;
   class substructure_classification;
   class substructure_lifting_data;

}


//! computes orbits using Schreier vectors

namespace orbits_schreier {


   // orbits
   class orbit_of_equations;
   class orbit_of_sets;
   class orbit_of_subspaces;

}




//! orbits on sets and subspaces using poset classification

namespace poset_classification {

   // poset_classification
   class classification_base_case;
   class extension;
   class orbit_based_testing;
   class orbit_tracer;
```

```
   class poset_classification;
   class poset_classification_control;
   class poset_classification_report_options;
   class poset_of_orbits;
   class poset_description;
   class poset_orbit_node;
   class poset_with_group_action;
   class upstep_work;

}

//! stabilizer of a set in a given action

namespace set_stabilizer {

   // set_stabilizer
   class compute_stabilizer;
   class stabilizer_orbits_and_types;
   class substructure_classifier;
   class substructure_stats_and_selection;

}

//! various solvers and tools to lift combinatorial structures

namespace solvers_package {

   // solver:
   class exact_cover_arguments;
   class exact_cover;

}


enum trace_result {
   found_automorphism,
   not_canonical,
   no_result_extension_not_found,
   no_result_fusion_node_installed,
   no_result_fusion_node_already_installed
};

enum find_isomorphism_result {
   fi_found_isomorphism,
   fi_not_isomorphic,
   fi_no_result
};
```

```
}}

#include "../layer4_classification/classify/classify.h"
#include "../layer4_classification/isomorph/isomorph.h"
#include "../layer4_classification/orbits/orbits.h"
#include "../layer4_classification/poset_classification/poset_classification.h"
#include "../layer4_classification/set_stabilizer/set_stabilizer.h"
#include "../layer4_classification/solver/l4_solver.h"



#endif /* ORBITER_SRC_LIB_CLASSIFICATION_CLASSIFICATION_H_ */
```

6.2 Classification based on a Relation

```
/*
 * classify.h
 *
 *  Created on: Aug 9, 2018
 *      Author: Anton Betten
 *
 * started:  September 20, 2007
 * pulled out of snakesandladders.h: Aug 9, 2018
 */


#ifndef ORBITER_SRC_LIB_CLASSIFICATION_CLASSIFY_CLASSIFY_H_
#define ORBITER_SRC_LIB_CLASSIFICATION_CLASSIFY_CLASSIFY_H_


namespace orbiter {
namespace layer4_classification {
namespace invariant_relations {



// ##############################################################################
// classification_step.cpp
// ##############################################################################

//! a single step classification of combinatorial objects


class classification_step {
public:
   actions::action *A; // do not free
   actions::action *A2; // do not free

   ring_theory::longinteger_object go;
   int max_orbits;
   int nb_orbits;
   orbit_node *Orbit; // [max_orbits]
   int representation_sz;
   long int *Rep; // [nb_orbits * representation_sz]

   classification_step();
   ~classification_step();
   void init(
         actions::action *A,
         actions::action *A2,
         int max_orbits, int representation_sz,
```

```
         ring_theory::longinteger_object &go,
         int verbose_level);
   data_structures_groups::set_and_stabilizer
      *get_set_and_stabilizer(
         int orbit_index,
         int verbose_level);
   void write_file(
         std::ofstream &fp, int verbose_level);
   void read_file(
         std::ifstream &fp,
         actions::action *A,
         actions::action *A2,
         ring_theory::longinteger_object &go,
         int verbose_level);
   void generate_source_code(
         std::string &fname_base, int verbose_level);
   void generate_source_code(
         std::ostream &ost, std::string &prefix, int verbose_level);
   long int *Rep_ith(int i);
   void print_group_orders();
   void print_summary(std::ostream &ost);
   void print_latex(std::ostream &ost,
         std::string &title,
         int f_print_stabilizer_gens,
      int f_has_print_function,
      void (*print_function)(std::ostream &ost, int i,
            classification_step *Step, void *print_function_data),
      void *print_function_data);

};

// ##############################################################################
// flag_orbits.cpp
// ##############################################################################

//! stores the set of flag orbits; related to the class classification_step


class flag_orbits {
public:
   actions::action *A; // do not free
   actions::action *A2; // do not free

   int nb_primary_orbits_lower;
   int nb_primary_orbits_upper;

   int upper_bound_for_number_of_traces;
```

```
   void (*func_to_free_received_trace)(
         void *trace_result, void *data, int verbose_level);
   void (*func_latex_report_trace)(
         std::ostream &ost, void *trace_result,
         void *data, int verbose_level);
   void *free_received_trace_data;

   int nb_flag_orbits;
   flag_orbit_node *Flag_orbit_node;
   int pt_representation_sz;
   long int *Pt; // [nb_flag_orbits * pt_representation_sz]


   flag_orbits();
   ~flag_orbits();
   void init(actions::action *A,
         actions::action *A2,
         int nb_primary_orbits_lower,
         int pt_representation_sz, int nb_flag_orbits,
         int upper_bound_for_number_of_traces,
         void (*func_to_free_received_trace)(
               void *trace_result, void *data, int verbose_level),
         void (*func_latex_report_trace)(
               std::ostream &ost, void *trace_result,
               void *data, int verbose_level),
         void *free_received_trace_data,
         int verbose_level);
   int find_node_by_po_so(int po, int so, int &idx,
      int verbose_level);
   void write_file(std::ofstream &fp, int verbose_level);
   void read_file(std::ifstream &fp,
         actions::action *A, actions::action *A2,
         int verbose_level);
   void print_latex(std::ostream &ost,
         std::string &title, int f_print_stabilizer_gens);

};

// ##############################################################################
// flag_orbit_node.cpp
// ##############################################################################

//! to represent a flag orbit; related to the class flag_orbits


class flag_orbit_node {
public:
```

```
   flag_orbits *Flag_orbits;

   int flag_orbit_index;

   int downstep_primary_orbit;
   int downstep_secondary_orbit;
   int downstep_orbit_len;
   int f_long_orbit;
   int upstep_primary_orbit;
   int upstep_secondary_orbit;
   int f_fusion_node;
   int fusion_with;
   int *fusion_elt;

   ring_theory::longinteger_object go;
   groups::strong_generators *gens;

   int nb_received;
   void **Receptacle; // [upper_bound_for_number_of_traces]

   flag_orbit_node();
   ~flag_orbit_node();
   void init(flag_orbits *Flag_orbits,
         int flag_orbit_index,
         int downstep_primary_orbit, int downstep_secondary_orbit,
         int downstep_orbit_len,
         int f_long_orbit,
         long int *pt_representation,
         groups::strong_generators *Strong_gens,
         int verbose_level);
   void receive_trace_result(
         void *trace_result, int verbose_level);
   void write_file(std::ofstream &fp, int verbose_level);
   void read_file(std::ifstream &fp, int verbose_level);
   void print_latex(flag_orbits *Flag_orbits,
         std::ostream &ost,
         int f_print_stabilizer_gens);

};

// ##############################################################################
// orbit_node.cpp
// ##############################################################################

//! to encode one group orbit, associated to the class classification_step

class orbit_node {
```

```
public:
   classification_step *C;
   int orbit_index;
   groups::strong_generators *gens;
   void *extra_data;

   orbit_node();
   ~orbit_node();
   void init(classification_step *C,
         int orbit_index,
         groups::strong_generators *gens,
         long int *Rep, void *extra_data,
         int verbose_level);
   void write_file(std::ofstream &fp, int verbose_level);
   void read_file(std::ifstream &fp, int verbose_level);
};

}}}


#endif /* ORBITER_SRC_LIB_CLASSIFICATION_CLASSIFY_CLASSIFY_H_ */
```

6.3 Isomorph Testing based on a Relation

```
// isomorph.h
//
// Anton Betten
//
// moved here from top_level.h: July 28, 2018
// top_level started:  September 23 2010
// based on global.h, which was taken from reader.h: 3/22/09



#ifndef ORBITER_SRC_LIB_TOP_LEVEL_ISOMORPH_ISOMORPH_H_
#define ORBITER_SRC_LIB_TOP_LEVEL_ISOMORPH_ISOMORPH_H_


namespace orbiter {
namespace layer4_classification {
namespace isomorph {

// ##############################################################################
// flag_orbit_folding.cpp
// ##############################################################################

//! folding of flag orbits during classification using class isomorph


class flag_orbit_folding {
public:

   isomorph *Iso;

   std::string event_out_fname;

   int current_flag_orbit; // previously orbit_no
      // Used in isomorph_testing:
      // The flag orbit we are currently testing.
      // In the end, this will become the representative of a
      // n e w isomorphism type

   representatives *Reps;

   int iso_nodes;




   int nb_open, nb_reps, nb_fused;
```

```
// for iso_test:


std::ofstream *fp_event_out;

// ToDo: bad use of global variables:
actions::action *AA;
actions::action *AA_perm;
actions::action *AA_on_k_subsets;

data_structures_groups::union_find *UF;
data_structures_groups::vector_ge *gens_perm;

int subset_rank;
int *subset;
long int *subset_witness;
long int *rearranged_set;
long int *rearranged_set_save;
long int *canonical_set;
long int *tmp_set;
int *Elt_transporter, *tmp_Elt, *Elt1, *transporter;

int cnt_minimal;
int NCK;
ring_theory::longinteger_object stabilizer_group_order;

int stabilizer_nb_generators;
int **stabilizer_generators;
   // int[stabilizer_nb_generators][size]


int *stabilizer_orbit;
int nb_is_minimal_called;
int nb_is_minimal;
int nb_sets_reached;


// temporary data used in identify_solution
int f_tmp_data_has_been_allocated;
long int *tmp_set1;
long int *tmp_set2;
long int *tmp_set3;
int *tmp_Elt1;
int *tmp_Elt2;
int *tmp_Elt3;
// temporary data used in trace_set_recursion
```

```
long int *trace_set_recursion_tmp_set1;
int *trace_set_recursion_Elt1;
int *trace_set_recursion_cosetrep;
// temporary data used in trace_set_recursion
long int *apply_fusion_tmp_set1;
int *apply_fusion_Elt1;
// temporary data used in find_extension
long int *find_extension_set1;
// temporary data used in make_set_smaller
long int *make_set_smaller_set;
int *make_set_smaller_Elt1;
int *make_set_smaller_Elt2;
// temporary data used in orbit_representative
int *orbit_representative_Elt1;
int *orbit_representative_Elt2;
// temporary data used in handle_automorphism
int *handle_automorphism_Elt1;
int *apply_isomorphism_tree_tmp_Elt;

flag_orbit_folding();
~flag_orbit_folding();
void init(isomorph *Iso, int verbose_level);
void isomorph_testing(
      int t0, int f_play_back,
      std::string &play_back_file_name,
   int f_implicit_fusion, int print_mod, int verbose_level);
   // calls do_iso_test
void do_iso_test(
      int t0, groups::sims *&Stab,
   int f_play_back, std::ifstream *play_back_file,
   int &f_eof, int print_mod,
   int f_implicit_fusion, int verbose_level);
int next_subset(int t0,
   int &f_continue, groups::sims *Stab, long int *data,
   int f_play_back, std::ifstream *play_back_file, int &f_eof,
   int verbose_level);
void process_rearranged_set(
      groups::sims *Stab, long int *data,
   int f_implicit_fusion, int verbose_level);
int is_minimal(int verbose_level);
void stabilizer_action_exit();
void stabilizer_action_init(int verbose_level);
   // Computes the permutations of the set
   // that are induced by the
   // generators for the stabilizer in AA
void stabilizer_action_add_generator(
      int *Elt, int verbose_level);
```

```
void print_statistics_iso_test(
      int t0, groups::sims *Stab);
int identify(long int *set, int f_implicit_fusion,
   int verbose_level);
int identify_database_is_open(long int *set,
   int f_implicit_fusion, int verbose_level);
void induced_action_on_set_basic(groups::sims *S,
   long int *set, int verbose_level);
void induced_action_on_set(groups::sims *S,
   long int *set, int verbose_level);
// Called by do_iso_test and print_isomorphism_types
// Creates the induced action on the set from the given action.
// The given action is gen->A2
// The induced action is computed to AA
// The set is in set[].
// Allocates a n e w union_find data structure and initializes it
// using the generators in S.
// Calls action::induced_action_by_restriction()
int handle_automorphism(long int *set,
      groups::sims *Stab, int *Elt, int verbose_level);
void print_isomorphism_types(int f_select,
   int select_first, int select_len,
   int verbose_level);
   // Calls print_set_function (if available)

int identify_solution_relaxed(
      long int *set, int *transporter,
   int f_implicit_fusion, int &orbit_no,
   int &f_failure_to_find_point,
   int verbose_level);
// returns the orbit number corresponding to
// the canonical version of set and the extension.
// Calls trace_set and find_extension_easy.
// Called from process_rearranged_set
int identify_solution(
      long int *set, int *transporter,
   int f_implicit_fusion, int &f_failure_to_find_point,
   int verbose_level);
   // returns the orbit number corresponding to
   // the canonical version of set and the extension.
   // Calls trace_set and find_extension_easy.
   // If needed, calls make_set_smaller
int trace_set(
      long int *canonical_set, int *transporter,
   int f_implicit_fusion, int &f_failure_to_find_point,
   int verbose_level);
   // returns the case number of the canonical set
```

```
   // (local orbit number)
   // Called from identify_solution and identify_solution_relaxed
   // calls trace_set_recursion
void make_set_smaller(int case_nb_local,
   long int *set, int *transporter, int verbose_level);
   // Called from identify_solution.
   // The goal is to produce a set that is lexicographically
   // smaller than the current starter.
   // To do this, we find an element that is less than
   // the largest element in the current starter.
   // There are two ways to find such an element.
   // Either, the set already contains such an element,
   // or one can produce such an element
   // by applying an element in the
   // stabilizer of the current starter.
int trace_set_recursion(int cur_level,
   int cur_node_global,
   long int *canonical_set, int *transporter,
   int f_implicit_fusion,
   int &f_failure_to_find_point, int verbose_level);
   // returns the node in the generator that corresponds
   // to the canonical_set.
   // Called from trace_set.
   // Calls trace_next_point and handle_extension.
int trace_next_point(int cur_level,
   int cur_node_global,
   long int *canonical_set, int *transporter,
   int f_implicit_fusion,
   int &f_failure_to_find_point, int verbose_level);
   // Called from trace_set_recursion
   // Calls trace_next_point_in_place
   // and (possibly) trace_next_point_database
   // Returns FALSE is the set becomes lexicographically smaller
int trace_next_point_database(int cur_level,
   int cur_node_global,
   long int *canonical_set, int *Elt_transporter,
   int verbose_level);
   // Returns FALSE is the set becomes lexicographically smaller
int handle_extension(int cur_level,
   int cur_node_global,
   long int *canonical_set, int *Elt_transporter,
   int f_implicit_fusion,
   int &f_failure_to_find_point, int verbose_level);
int handle_extension_database(int cur_level,
   int cur_node_global,
   long int *canonical_set, int *Elt_transporter,
   int f_implicit_fusion,
```

```
   int &f_failure_to_find_point,
   int verbose_level);
int handle_extension_tree(int cur_level,
   int cur_node_global,
   long int *canonical_set, int *Elt_transporter,
   int f_implicit_fusion,
   int &f_failure_to_find_point,
   int verbose_level);
// Returns next_node_global at level cur_level + 1.
void apply_isomorphism_database(int cur_level,
   int cur_node_global,
   int current_extension, long int *canonical_set,
   int *Elt_transporter, int ref,
   int verbose_level);
void apply_isomorphism_tree(int cur_level,
   int cur_node_global,
   int current_extension, long int *canonical_set,
   int *Elt_transporter,
   int verbose_level);

void handle_event_files(int nb_event_files,
   const char **event_file_name, int verbose_level);
void read_event_file(
      const char *event_file_name,
   int verbose_level);
void skip_through_event_file(std::ifstream &f,
   int verbose_level);
void skip_through_event_file1(std::ifstream &f,
   int case_no, int orbit_no, int verbose_level);
void event_file_completed_cases(
      const char *event_file_name,
   int &nb_completed_cases, int *completed_cases,
   int verbose_level);
void event_file_read_case(const char *event_file_name,
   int case_no, int verbose_level);
void event_file_read_case1(std::ifstream &f,
   int case_no, int verbose_level);
int next_subset_play_back(int &subset_rank,
      std::ifstream *play_back_file,
   int &f_eof, int verbose_level);

void write_classification_matrix(int verbose_level);
void write_classification_graph(int verbose_level);
void decomposition_matrix(int verbose_level);
void compute_down_link(
      int *&down_link, int verbose_level);
void probe(int flag_orbit, int subset_rk,
```

```
      int f_implicit_fusion, int verbose_level);
   void test_compute_stabilizer(int verbose_level);
   void test_memory(int verbose_level);
   void test_edges(int verbose_level);
   int test_edge(int n1, long int *subset1,
      int *transporter, int verbose_level);
   void compute_Ago_Ago_induced(
         ring_theory::longinteger_object *&Ago,
         ring_theory::longinteger_object *&Ago_induced,
         int verbose_level);
   void get_orbit_transversal(
         data_structures_groups::orbit_transversal *&T,
      int verbose_level);
   void compute_stabilizer(
         groups::sims *&Stab, int verbose_level);
   void iso_test_init(int verbose_level);
   void iso_test_init2(int verbose_level);

};



// ##############################################################################
// isomorph_arguments.cpp
// ##############################################################################

//! auxiliary class for class isomorph


class isomorph_arguments {
public:
   int f_init_has_been_called;

   int f_use_database_for_starter;
   int f_implicit_fusion;

   int f_build_db;

   int f_read_solutions;

   int f_list_of_cases;
   std::string list_of_cases_fname;

   //int f_read_solutions_from_clique_finder;
   //int f_read_solutions_from_clique_finder_list_of_cases;
   //std::string fname_list_of_cases;
   int f_read_solutions_after_split;
```

```
int read_solutions_split_m;

int f_read_statistics_after_split;
//int read_statistics_split_m;

int f_recognize;
std::string recognize_label;

int f_compute_orbits;
int f_isomorph_testing;
int f_classification_graph;
int f_event_file; // -e <event file> option
std::string event_file_name;
int print_mod;

int f_isomorph_report;


int f_subset_orbits;
int f_subset_orbits_file;
std::string subset_orbits_fname;
int f_eliminate_graphs_if_possible;
int f_down_orbits;

int f_prefix_iso;
std::string prefix_iso;

actions::action *A;
actions::action *A2;
poset_classification::poset_classification *gen;
int target_size;
poset_classification::poset_classification_control *Control;

int f_prefix_with_directory;
std::string prefix_with_directory;

int f_prefix_classify;
std::string prefix_classify;

int f_solution_prefix;
std::string solution_prefix;

int f_base_fname;
std::string base_fname;

solvers_package::exact_cover_arguments *ECA;
```

```
   void (*callback_report)(isomorph *Iso, void *data,
      int verbose_level);
   void (*callback_subset_orbits)(isomorph *Iso, void *data,
      int verbose_level);
   void *callback_data;

   int f_has_final_test_function;
   int (*final_test_function)(long int *data, int sz,
      void *final_test_data, int verbose_level);
   void *final_test_data;

   isomorph_arguments();
   ~isomorph_arguments();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();
   void init(actions::action *A,
         actions::action *A2,
         poset_classification::poset_classification *gen,
      int target_size,
      poset_classification::poset_classification_control *Control,
      solvers_package::exact_cover_arguments *ECA,
      void (*callback_report)(
            isomorph *Iso, void *data,
         int verbose_level),
      void (*callback_subset_orbits)(
            isomorph *Iso, void *data,
         int verbose_level),
      void *callback_data,
      int verbose_level);
   //void execute(int verbose_level);

};

//! auxiliary class to pass case specific data to the function isomorph_worker

struct isomorph_worker_data {
   long int *the_set;
   int set_size;
   void *callback_data;
};




// ##############################################################################
// isomorph_global.cpp
```

```
// ##############################################################################


//! auxiliary class for class isomorph


class isomorph_global {

public:

   actions::action *A_base;
   actions::action *A;

   poset_classification::poset_classification *gen;


   isomorph_global();
   ~isomorph_global();
   void init(
         actions::action *A_base,
         actions::action *A,
         poset_classification::poset_classification *gen,
         int verbose_level);
   void read_statistic_files(
      int size, std::string &prefix_classify,
      std::string &prefix, int level,
      std::string *fname, int nb_files,
      int verbose_level);
   void init_solutions_from_memory(
      int size, std::string &prefix_classify,
      std::string &prefix_iso, int level,
      long int **Solutions, int *Nb_sol, int verbose_level);
   void classification_graph(
      int size, std::string &prefix_classify,
      std::string &prefix_iso, int level,
      int verbose_level);
   void identify(
      int size, std::string &prefix_classify,
      std::string &prefix_iso, int level,
      int identify_nb_files,
      std::string *fname, int *Iso_type,
      int f_save, int verbose_level);
   void identify_table(
      int size, std::string &prefix_classify,
      std::string &prefix_iso, int level,
      int nb_rows, long int *Table, int *Iso_type,
      int verbose_level);
```

```
   void worker(
      int size,
      std::string &prefix_classify,
      std::string &prefix_iso,
      void (*work_callback)(
            isomorph *Iso, void *data, int verbose_level),
      void *work_data,
      int level, int verbose_level);
   void compute_down_orbits(
      int size,
      std::string &prefix_classify,
      std::string &prefix,
      int level, int verbose_level);
   void compute_down_orbits_for_isomorphism_type(
      isomorph *Iso, int orbit,
      int &cnt_orbits, int &cnt_special_orbits,
      int *&special_orbit_identify, int verbose_level);
   void report_data_in_source_code_inside_tex(
         isomorph &Iso,
         const char *prefix,
         char *label_of_structure_plural, std::ostream &f,
         int verbose_level);
   void report_data_in_source_code_inside_tex_with_selection(
         isomorph &Iso, const char *prefix,
         char *label_of_structure_plural, std::ostream &fp,
         int selection_size, int *selection,
         int verbose_level);

};


// ##############################################################################
// isomorph_worker.cpp
// ##############################################################################

//! main class to run a classification algorithm through class isomorph and class
 isomorph_global


class isomorph_worker {
public:

   isomorph_arguments *Isomorph_arguments;

   isomorph_global *Isomorph_global;

   isomorph *Iso;
```

```
   isomorph_worker();
   ~isomorph_worker();
   void init(
         isomorph_arguments *Isomorph_arguments,
         actions::action *A_base,
         actions::action *A,
         poset_classification::poset_classification *gen,
         int size, int level,
         int verbose_level);
   void execute(
         isomorph_arguments *Isomorph_arguments,
         int verbose_level);
   void build_db(int verbose_level);
   void read_solutions(int verbose_level);
   void compute_orbits(int verbose_level);
   void isomorph_testing(int verbose_level);
   void isomorph_report(int verbose_level);
   void report(std::ostream &ost, int verbose_level);
   void recognize(std::string &label, int verbose_level);

};




// ##############################################################################
// isomorph.cpp
// ##############################################################################

//! classification of combinatorial objects using subobjects


class isomorph {
public:
   int size; // size of one solution
   int level; // size of one subobject

   std::string prefix;
   std::string prefix_invariants;
   std::string prefix_tex;



   actions::action *A_base;
      // A Betten 3/18/2013
      // the action in which we represent the group
```

```
   // do not free
actions::action *A;
   // the action in which we act on the set
   // do not free


// the classification of substructures

substructure_classification *Sub;

// lifting data:

substructure_lifting_data *Lifting;


flag_orbit_folding *Folding;



void (*print_set_function)(isomorph *Iso,
   int iso_cnt, groups::sims *Stab, groups::schreier &Orb,
   long int *data, void *print_set_data, int verbose_level);
void *print_set_data;


// some statistics:
int nb_times_make_set_smaller_called;

isomorph();
~isomorph();
void init(std::string &prefix,
      actions::action *A_base,
      actions::action *A,
      poset_classification::poset_classification *gen,
   int size, int level,
   int f_use_database_for_starter,
   int f_implicit_fusion, int verbose_level);
void print_node_local(int level, int node_local);
void print_node_global(int level, int node_global);
void init_high_level(
      actions::action *A,
      poset_classification::poset_classification *gen,
   int size,
   std::string &prefix_classify,
   std::string &prefix,
   int level, int verbose_level);
void induced_action_on_set_and_kernel(
```

```
         std::ostream &file,
         actions::action *A,
         groups::sims *Stab, int size, long int *set,
      int verbose_level);
   void read_everything_including_classification(
         std::string &prefix_classify, int verbose_level);


};




// ##############################################################################
// representatives.cpp
// ##############################################################################

//! auxiliary class for class isomorph



class representatives {
public:
   actions::action *A;

   std::string prefix;
   std::string fname_rep;
   std::string fname_stabgens;
   std::string fname_fusion;
   std::string fname_fusion_ge;



   // flag orbits:
   int nb_objects;
   int *fusion; // [nb_objects]
      // fusion[i] == -2 means that the flag orbit i
      // has not yet been processed by the
      // isomorphism testing procedure.
      // fusion[i] = i means that flag orbit [i]
      // in an orbit representative
      // Otherwise, fusion[i] is an earlier flag_orbit,
      // and handle[i] is a group element that maps
      // to it
   int *handle; // [nb_objects]
      // handle[i] is only relevant if fusion[i] != i,
      // i.e., if flag orbit i is not a representative
```

```
      // of an isomorphism type.
      // In this case, handle[i] is the (handle of a)
      // group element moving flag orbit i to flag orbit fusion[i].


   // classified objects:
   int count;
   long int *rep; // [count]
   groups::sims **stab; // [count]



   //char *elt;
   int *Elt1;
   int *tl; // [A->base_len]

   int nb_open;
   int nb_reps;
   int nb_fused;


   representatives();
   ~representatives();
   void init(
         actions::action *A,
         int nb_objects,
         std::string &prefix,
         int verbose_level);
   void write_fusion(int verbose_level);
   void read_fusion(int verbose_level);
   void write_representatives_and_stabilizers(
         int verbose_level);
   void read_representatives_and_stabilizers(
         int verbose_level);
   void get_stabilizer(isomorph *Iso, int idx,
         groups::strong_generators *&SG,
         int verbose_level);
   void save(int verbose_level);
   void load(int verbose_level);
   void calc_fusion_statistics();
   void print_fusion_statistics();
};


// ##############################################################################
// substructure_classification.spp
// ##############################################################################
```

```
//! the classification of substructures


class substructure_classification {
public:

   isomorph *Iso;


   int f_use_database_for_starter;
   int depth_completed;
   int f_use_implicit_fusion;




   std::string fname_db_level_ge;

   std::string fname_db_level;
   std::string fname_db_level_idx1;
   std::string fname_db_level_idx2;


   int nb_starter;
      // the number of orbits at ´level´,
      // previously called nb_cases


      // solution_first and solution_len are initialized in
      // isomorph_files.cpp
      // isomorph::init_solutions
      // isomorph::count_solutions_from_clique_finder
      // isomorph::count_solutions

      // they are written to file in
      // isomorph::write_solution_first_and_len()


   poset_classification::poset_classification *gen;
      // do not free


   // database access:
   typed_objects::database *D1, *D2;
   std::string fname_ge1;
   std::string fname_ge2;
```

```
std::ifstream *fp_ge1;
std::ifstream *fp_ge2;

// pointer only, do not free:
typed_objects::database *DB_level;
std::ifstream *fp_ge; // either fg_ge1 or fp_ge2

substructure_classification();
~substructure_classification();
void init(isomorph *Iso,
      poset_classification::poset_classification *gen,
      int f_use_database_for_starter,
      int f_implicit_fusion,
      int verbose_level);
void read_data_files_for_starter(int level,
      std::string &prefix, int verbose_level);
   // Calls gen->read_level_file_binary
   // for all levels i from 0 to level
   // Uses letter a files for i from 0 to level - 1
   // and letter b file for i = level.
   // If gen->f_starter is TRUE,
   // we start from i = gen->starter_size instead.
   // Finally, it computes nb_starter.
void compute_nb_starter(int level, int verbose_level);
void print_node_local(int level, int node_local);
void print_node_global(int level, int node_global);

void setup_and_open_level_database(int verbose_level);
// Called from do_iso_test, identify and test_hash
// (Which are all in isomorph_testing.cpp)
// Calls init_DB for D and D.open.
// Calls init_DB_level for D1 and D2 and D1->open and D2->open.
// Calls fopen for fp_ge1 and fp_ge2.
void close_level_database(int verbose_level);
// Closes D1, D2, fp_ge1, fp_ge2.
void prepare_database_access(
      int cur_level, int verbose_level);
// sets D to be D1 or D2, depending on cur_level
// Called from
// load_strong_generators
// trace_next_point_database
void find_extension_easy(long int *set,
   int case_nb, int &idx,
   int &f_found, int verbose_level);
   // returns TRUE if found, FALSE otherwise
   // Called from identify_solution
   // Linear search through all solutions at a given starter.
```

```
      // calls load solution for each of the solutions
      // stored with the case and compares the vectors.
   int find_extension_search_interval(long int *set,
      int first, int len, int &idx,
      int f_btree_idx, int btree_idx,
      int f_through_hash, int verbose_level);
   int find_extension_easy_old(long int *set,
      int case_nb, int &idx, int verbose_level);
   void find_extension_easy_new(long int *set,
      int case_nb, int &idx, int &f_found, int verbose_level);
   int open_database_and_identify_object(long int *set,
      int *transporter,
      int f_implicit_fusion, int verbose_level);
   void init_DB_level(
         layer2_discreta::typed_objects::database &D, int level,
      int verbose_level);
   void create_level_database(int level, int verbose_level);
   void load_strong_generators(int cur_level,
      int cur_node_local,
      data_structures_groups::vector_ge &gens,
      ring_theory::longinteger_object &go,
      int verbose_level);
      // Called from compute_stabilizer and
      // from orbit_representative
   void load_strong_generators_tree(int cur_level,
      int cur_node_local,
      data_structures_groups::vector_ge &gens,
      ring_theory::longinteger_object &go,
      int verbose_level);
   void load_strong_generators_database(int cur_level,
      int cur_node_local,
      data_structures_groups::vector_ge &gens,
      ring_theory::longinteger_object &go,
      int verbose_level);
      // Reads node cur_node_local (local index)
      // from database D through btree 0
      // Reads generators from file fp_ge

};



// ##############################################################################
// substructure_lifting_data.spp
// ##############################################################################

//! the lifting of the representatives of substructure isomorphism types
```

```
class substructure_lifting_data {
public:

   isomorph *Iso;

   std::string fname_flag_orbits;
   std::string fname_stab_orbits;
   std::string fname_case_len;
   std::string fname_statistics;
   std::string fname_hash_and_datref;
   std::string fname_db1;
   std::string fname_db2;
   std::string fname_db3;
   std::string fname_db4;
   std::string fname_db5;

   std::string fname_orbits_of_stabilizer_csv;


   int nb_flag_orbits;
   // Number of flag orbits.
   // Overall number of orbits of stabilizers
   // of starters on the solutions
   // computed in orbits_of_stabilizer,
   // which in turn calls orbits_of_stabilizer_case
   // for each starter
   // orbits_of_stabilizer_case takes the
   // strong generators from the
   // generator data structure
   // For computing the orbits, induced_action_on_sets
   // is used to establish an action on sets.

   int N;
      // the number of solutions,
      // computed in init_cases_from_file
      // or read from file in read_case_len



   int *solution_first;
      // [nb_starter + 1] the beginning of solutions
      // belonging to a given starter
      // previously called case_first
   int *solution_len;
      // [nb_starter + 1] the number of solutions
```

```
   // belonging to a given starter
   // previously called case_len



int *starter_number;
   // [N]  starter_number[i] = j means that
   // solution i belongs to starter j
   // previously called case_number

int *orbit_fst;
   // [nb_flag_orbits + 1]
   // orbit_fst[i] is the beginning of solutions
   // associated to the i-th flag orbit
   // in the sorted list of solutions
   // allocated in isomorph::read_orbit_data()

int *orbit_len;
   // [nb_orbits]
   // orbit_len[i] is the length of the i-th flag orbit
   // allocated in isomorph::read_orbit_data()


int *orbit_number;
   // [N]
   // orbit_number[i] is the flag orbit
   // containing the i-th solution
   // in the original labeling
   // allocated in isomorph::read_orbit_data()


int *orbit_perm;
   // [N]
   // orbit_perm[i] is the original label
   // of the i-th solution in the ordered list
   // we often see id = orbit_perm[orbit_fst[orbit_no]];
   // this is the index of the first solution
   // associated to flag orbit orbit_no, for instance
   // allocated in isomorph::read_orbit_data()

int *orbit_perm_inv;
   // [N]
   // orbit_perm_inv is the inverse of orbit_perm
   // allocated in isomorph::read_orbit_data()


int *schreier_vector; // [N]
```

```
   // allocated in isomorph::read_orbit_data()
int *schreier_prev; // [N]
   // allocated in isomorph::read_orbit_data()



// orbit_fst[nb_flag_orbits + 1]
// orbit_len[nb_flag_orbits]
// orbit_number[N]
// orbit_perm[N]
// orbit_perm_inv[N]
// schreier_vector[N]
// schreier_prev[N]

int f_use_table_of_solutions;
long int *table_of_solutions; // [N * size]

// added Dec 25, 2012:
// computed in isomorph.cpp isomorph::orbits_of_stabilizer
// these variables are not used ?? (Oct 30, 2014)
// They are used, for instance in isomorph_testing.cpp
// isomorph::write_classification_graph (May 3, 2015)
int *flag_orbit_fst;
   // [nb_starter + 1]
   // the beginning of flag orbits
   // belonging to a given starter
int *flag_orbit_len;
   // [nb_starter]
   // the number of flag orbits belonging to a given starter

// from the summary file (and only used in isomorph_files.cpp)
int *stats_nb_backtrack;
   // [nb_starter]
int *stats_nb_backtrack_decision;
   // [nb_starter]
int *stats_graph_size;
   // [nb_starter]
int *stats_time;
   // [nb_starter]


typed_objects::Vector *v; // [1]
typed_objects::database *DB_sol;
long int *id_to_datref;
long int *id_to_hash;
long int *hash_vs_id_hash; // sorted
long int *hash_vs_id_id;
```

```
substructure_lifting_data();
~substructure_lifting_data();
void init(isomorph *Iso, int verbose_level);
void write_solution_first_and_len(int verbose_level);
void read_solution_first_and_len(int verbose_level);
void init_starter_number(int verbose_level);
void init_solution(int verbose_level);
void load_table_of_solutions(int verbose_level);
void list_solutions_by_starter(int verbose_level);
void list_solutions_by_orbit(int verbose_level);
void orbits_of_stabilizer(int verbose_level);
void orbits_of_stabilizer_case(int the_case,
      data_structures_groups::vector_ge &gens,
      int verbose_level);
void orbit_representative(int i, int &i0,
   int &orbit, int *transporter, int verbose_level);
   // slow because it calls load_strong_generators
void test_orbit_representative(int verbose_level);
void test_identify_solution(int verbose_level);
void setup_and_open_solution_database(int verbose_level);
void setup_and_create_solution_database(int verbose_level);
void close_solution_database(int verbose_level);
void init_DB_sol(int verbose_level);
   // We assume that the starter is of size 5 and that
   // fields 3-8 are the starter
void add_solution_to_database(long int *data,
   int nb, int id, int no,
   int nb_solutions, long int h, uint_4 &datref,
   int print_mod, int verbose_level);
void load_solution(
      int id, long int *data, int verbose_level);
void load_solution_by_btree(int btree_idx,
   int idx, int &id, long int *data);
void count_solutions(
      int nb_files,
      std::string *fname,
      int *List_of_cases, int *&Nb_sol_per_file,
      int f_get_statistics,
      int f_has_final_test_function,
      int (*final_test_function)(
            long int *data, int sz,
            void *final_test_data, int verbose_level),
      void *final_test_data,
      int verbose_level);
void add_solutions_to_database(
      long int *Solutions,
```

```
   int the_case, int nb_solutions, int nb_solutions_total,
   int print_mod, int &no,
   int verbose_level);
void init_solutions(long int **Solutions,
   int *Nb_sol, int verbose_level);
// Solutions[nb_starter], Nb_sol[nb_starter]
void count_solutions_from_clique_finder_case_by_case(
      int nb_files,
   long int *list_of_cases, std::string *fname,
   int verbose_level);
void count_solutions_from_clique_finder(
      int nb_files,
      std::string *fname,
   int verbose_level);
void read_solutions_from_clique_finder_case_by_case(
      int nb_files,
   long int *list_of_cases, std::string *fname,
   int verbose_level);
void read_solutions_from_clique_finder(
      int nb_files,
      std::string *fname,
      int *substructure_case_number, int *Nb_sol_per_file,
      int verbose_level);
void build_up_database(
      int nb_files,
      std::string *fname,
   int f_has_final_test_function,
   int (*final_test_function)(
         long int *data, int sz,
      void *final_test_data, int verbose_level),
   void *final_test_data,
   int verbose_level);
void get_statistics(int nb_files,
      std::string *fname, int *List_of_cases,
      int verbose_level);
void write_statistics();
void evaluate_statistics(int verbose_level);
void write_starter_nb_orbits(int verbose_level);
void read_starter_nb_orbits(int verbose_level);
void write_hash_and_datref_file(int verbose_level);
   // Writes the file ´fname_hash_and_datref´
   // containing id_to_hash[] and id_to_datref[]
void read_hash_and_datref_file(int verbose_level);
   // Reads the file ´fname_hash_and_datref´
   // containing id_to_hash[] and id_to_datref[]
   // Also initializes hash_vs_id_hash and hash_vs_id_id
   // Called from init_solution
```

```
   void print_hash_vs_id();


   void write_orbit_data(int verbose_level);
      // Writes the file ´fname_staborbits´
   void read_orbit_data(int verbose_level);
      // Reads from the file ´fname_staborbits´
      // Reads nb_orbits, N,
      // orbit_fst[nb_flag_orbits + 1]
      // orbit_len[nb_flag_orbits]
      // orbit_number[N]
      // orbit_perm[N]
      // schreier_vector[N]
      // schreier_prev[N]
      // and computed orbit_perm_inv[N]
   void test_hash(int verbose_level);
   void id_to_datref_allocate(int verbose_level);

};


}}}



#endif /* ORBITER_SRC_LIB_TOP_LEVEL_ISOMORPH_ISOMORPH_H_ */
```

6.4 Orbit Algorithms

```
// orbits.h
//
// Anton Betten
//
// moved here from top_level.h: July 28, 2018
// top_level started:  September 23 2010
// based on global.h, which was taken from reader.h: 3/22/09


#ifndef ORBITER_SRC_LIB_TOP_LEVEL_ORBITS_ORBITS_H_
#define ORBITER_SRC_LIB_TOP_LEVEL_ORBITS_ORBITS_H_




namespace orbiter {
namespace layer4_classification {
namespace orbits_schreier {



// ##############################################################################
// orbit_of_equations.cpp
// ##############################################################################


//! orbit of homogeneous equations using a Schreier tree


class orbit_of_equations {
public:
   actions::action *A;
   induced_actions::action_on_homogeneous_polynomials *AonHPD;
   field_theory::finite_field *F;
   groups::strong_generators *SG;

   int nb_monomials;
   int sz; // = 1 + nb_monomials
   int sz_for_compare; // = 1 + nb_monomials
   int *data_tmp; // [sz]

   int position_of_original_object;
   int allocation_length;
   int used_length;

   int **Equations; // [allocation_length][sz]
```

```
int *prev; // [allocation_length]
int *label; // [allocation_length]

int f_has_print_function;
void (*print_function)(int *object,
      int sz, void *print_function_data);
void *print_function_data;

int f_has_reduction;
void (*reduction_function)(int *object,
      void *reduction_function_data);
void *reduction_function_data;


orbit_of_equations();
~orbit_of_equations();
void init(
      actions::action *A,
      field_theory::finite_field *F,
      induced_actions::action_on_homogeneous_polynomials
         *AonHPD,
   groups::strong_generators *SG,
   int *coeff_in,
   int verbose_level);
void map_an_equation(
      int *object_in, int *object_out,
   int *Elt, int verbose_level);
void print_orbit();
void compute_orbit(int *coeff, int verbose_level);
void reallocate(
      int *&Q, int Q_len, int verbose_level);
void get_transporter(
      int idx, int *transporter, int verbose_level);
   // transporter is an element which maps
   // the orbit representative to the given subspace.
void get_random_schreier_generator(
      int *Elt, int verbose_level);
void get_canonical_form(
      int *canonical_equation,
      int *transporter_to_canonical_form,
      groups::strong_generators
         *&gens_stab_of_canonical_equation,
      ring_theory::longinteger_object
         &full_group_order,
      int verbose_level);
groups::strong_generators *stabilizer_orbit_rep(
      ring_theory::longinteger_object &full_group_order,
```

```
         int verbose_level);
   void stabilizer_orbit_rep_work(
         actions::action *default_action,
         ring_theory::longinteger_object &go,
         groups::sims *&Stab, int verbose_level);
      // this function allocates a sims structure into Stab.
   groups::strong_generators *stabilizer_any_point(
         ring_theory::longinteger_object &full_group_order,
         int idx,
      int verbose_level);
   int search_equation(
         int *eqn, int &idx, int verbose_level);
   int search_data(
         int *data, int &idx, int verbose_level);
   void save_csv(
         std::string &fname, int verbose_level);
};


// ##############################################################################
// orbit_of_sets.cpp
// ##############################################################################





//! orbit of sets using a Schreier tree, used in packing::make_spread_table


class orbit_of_sets {
public:
   actions::action *A;
   actions::action *A2;
   data_structures_groups::vector_ge *gens;
   long int *set;
      // the set whose orbit we want to compute;
      // it is of size ´sz´
   int sz;

   int position_of_original_set;
      // = 0; never changes
   int allocation_length;
      // number of entries allocated in Sets
   int old_length;
   int used_length;
```

```
   // number of sets currently stored in Sets
long int **Sets;
   // the sets are stored in the order in which they
   // are discovered and added to the tree
int *Extra;
   // [allocation_length * 2]
   // Extra[i * 2 + 0] is the index of the ancestor node of node i.
   // Extra[i * 2 + 1] is the label of the generator that maps
   // the ancestor of node i to node i.
   // Here, node i means the set in Sets[i].
   // Node 0 is the root node, i.e. the set in ´set´.
int *cosetrep; // the result of coset_rep()
int *cosetrep_tmp; // temporary storage for coset_rep()

std::multimap<uint32_t, int> Hashing;
   // we store the pair (hash, idx)
   // where hash is the hash value of the set and idx is the
   // index in the table Sets where the set is stored.
   //
   // we use a multimap because the hash values are not unique
   // it happens that two sets have the same hash value.
   // map cannot handle that.

orbit_of_sets();
~orbit_of_sets();
void init(actions::action *A,
      actions::action *A2,
      long int *set, int sz,
      data_structures_groups::vector_ge *gens,
      int verbose_level);
void compute(int verbose_level);
void setup_root_node(
      long int *Q, int &Q_len, int verbose_level);
void reallocate(
      long int *&Q, int Q_len, int verbose_level);
void dump_tables_of_hash_values();
void get_table_of_orbits(
      long int *&Table, int &orbit_length,
   int &set_size, int verbose_level);
void get_table_of_orbits_and_hash_values(
      long int *&Table,
      int &orbit_length,
      int &set_size, int verbose_level);
void make_table_of_coset_reps(
      data_structures_groups::vector_ge *&Coset_reps,
      int verbose_level);
void coset_rep(int j);
```

```
      // result is in cosetrep
      // determines an element in the group
      // that moves the orbit representative
   // to the j-th element in the orbit.
   void get_orbit_of_points(
         std::vector<long int> &Orbit,
         int verbose_level);
   void get_prev(
         std::vector<int> &Prev,
         int verbose_level);
   void get_label(
         std::vector<int> &Label,
         int verbose_level);
};


// ##############################################################################
// orbit_of_subspaces.cpp
// ##############################################################################



//! orbit of subspaces using a Schreier tree



class orbit_of_subspaces {
public:
   actions::action *A;
   actions::action *A2;
   field_theory::finite_field *F;
   data_structures_groups::vector_ge *gens;
   int f_lint;
   int k;
   int n;
   int kn;
   int sz; // = 1 + k + kn
   //int sz_for_compare; // = 1 + k + kn
   int f_has_desired_pivots;
   int *desired_pivots; // [k]
   int *subspace_by_rank; // [k]
   long int *subspace_by_rank_lint; // [k]
   int *data_tmp; // [sz]
   int *Mtx1;
   int *Mtx2;
   int *Mtx3;
```

```
int f_has_rank_functions;
void *rank_unrank_data;
int (*rank_vector_callback)(int *v, int n,
   void *data, int verbose_level);
long int (*rank_vector_lint_callback)(int *v, int n,
   void *data, int verbose_level);
void (*unrank_vector_callback)(int rk, int *v,
   int n, void *data, int verbose_level);
void (*unrank_vector_lint_callback)(long int rk, int *v,
   int n, void *data, int verbose_level);
void (*compute_image_of_vector_callback)(int *v, int *w,
   int *Elt, void *data, int verbose_level);
void *compute_image_of_vector_callback_data;

int position_of_original_subspace;
int allocation_length;
int old_length;
int used_length;
int **Subspaces;
long int **Subspaces_lint;
int *prev;
int *label;

std::multimap<uint32_t, int> Hashing;
   // we store the pair (hash, idx)
   // where hash is the hash value of the set and idx is the
   // index in the table Sets where the set is stored.
   //
   // we use a multimap because the hash values are not unique
   // it happens that two sets have the same hash value.
   // map cannot handle that.


orbit_of_subspaces();
~orbit_of_subspaces();
void init(actions::action *A,
      actions::action *A2,
      field_theory::finite_field *F,
   int *subspace, int k, int n,
   int f_has_desired_pivots, int *desired_pivots,
   int f_has_rank_functions, void *rank_unrank_data,
   int (*rank_vector_callback)(int *v, int n,
      void *data, int verbose_level),
   void (*unrank_vector_callback)(int rk, int *v,
      int n, void *data, int verbose_level),
   void (*compute_image_of_vector_callback)(int *v,
      int *w, int *Elt, void *data, int verbose_level),
```

```
   void *compute_image_of_vector_callback_data,
   data_structures_groups::vector_ge *gens,
   int verbose_level);
void init_lint(
      actions::action *A,
      actions::action *A2,
      field_theory::finite_field *F,
   long int *subspace_by_rank, int k, int n,
   int f_has_desired_pivots, int *desired_pivots,
   int f_has_rank_functions, void *rank_unrank_data,
   long int (*rank_vector_lint_callback)(int *v, int n,
         void *data, int verbose_level),
   void (*unrank_vector_lint_callback)(long int rk, int *v, int n,
         void *data, int verbose_level),
   void (*compute_image_of_vector_callback)(int *v, int *w,
         int *Elt, void *data, int verbose_level),
   void *compute_image_of_vector_callback_data,
   data_structures_groups::vector_ge *gens,
   int verbose_level);
int rank_vector(int *v, int verbose_level);
long int rank_vector_lint(int *v, int verbose_level);
void unrank_vector(
      int rk, int *v, int verbose_level);
void unrank_vector_lint(
      long int rk, int *v, int verbose_level);
void unrank_subspace(
      int subspace_idx,
      int *subspace_basis, int verbose_level);
void rank_subspace(
      int *subspace_basis, int verbose_level);
uint32_t hash_subspace();
void unrank(
      int *rk, int *subspace_basis, int verbose_level);
void unrank_lint(
      long int *rk, int *subspace_basis, int verbose_level);
void rank(
      int *rk, int *subspace_basis, int verbose_level);
void rank_lint(
      long int *rk, int *subspace_basis, int verbose_level);
void rref(int *subspace, int verbose_level);
void rref_and_rank(
      int *subspace, int *rk, int verbose_level);
void rref_and_rank_lint(
      int *subspace, long int *rk, int verbose_level);
void map_a_subspace(
      int *basis, int *image_basis, int *Elt,
   int verbose_level);
```

```
   void print_orbit();
   int rank_hash_and_find(
         int *subspace,
         int &idx, uint32_t &h, int verbose_level);
   void compute(int verbose_level);
   void get_transporter(
         int idx, int *transporter, int verbose_level);
      // transporter is an element which maps the orbit
      // representative to the given subspace.
   int find_subspace(
         int *subspace_ranks,
         int &idx, int verbose_level);
   int find_subspace_lint(
         long int *subspace_ranks,
         int &idx, int verbose_level);
   void get_random_schreier_generator(
         int *Elt, int verbose_level);
   groups::strong_generators *stabilizer_orbit_rep(
         ring_theory::longinteger_object &full_group_order,
         int verbose_level);
   void compute_stabilizer(
         actions::action *default_action,
         ring_theory::longinteger_object &go,
         groups::sims *&Stab, int verbose_level);
      // this function allocates a sims structure into Stab.
};


}}}


#endif /* ORBITER_SRC_LIB_TOP_LEVEL_ORBITS_ORBITS_H_ */
```

6.5 Poset Classification

```
/*
 * poset_classification.h
 *
 *  Created on: Aug 9, 2018
 *      Author: Anton Betten
 *
 * started:  September 20, 2007
 * pulled out of snakesandladders.h: Aug 9, 2018
 */


#ifndef ORBITER_SRC_LIB_CLASSIFICATION_POSET_CLASSIFICATION_POSET_CLASSIFICATION_
H_
#define ORBITER_SRC_LIB_CLASSIFICATION_POSET_CLASSIFICATION_POSET_CLASSIFICATION_
H_



namespace orbiter {
namespace layer4_classification {
namespace poset_classification {


// ##############################################################################
// classification_base_case.cpp
// ##############################################################################



//! represents a known classification with constructive recognition, to be used a
s base case for poset_classification



class classification_base_case {

public:
   poset_with_group_action *Poset;

   int size;
   long int *orbit_rep; // [size]
   groups::strong_generators *Stab_gens;
   long int *live_points;
   int nb_live_points;
   void *recognition_function_data;
   int (*recognition_function)(long int *Set, int len, int *Elt,
```

```
      void *data, int verbose_level);
   int *Elt;

   classification_base_case();
   ~classification_base_case();
   void init(
         poset_with_group_action *Poset,
         int size, long int *orbit_rep,
         long int *live_points, int nb_live_points,
         groups::strong_generators *Stab_gens,
         void *recognition_function_data,
         int (*recognition_function)(long int *Set, int len,
               int *Elt, void *data, int verbose_level),
         int verbose_level);
   int invoke_recognition(long int *Set, int len,
         int *Elt, int verbose_level);
};

// ##############################################################################
// extension.cpp
// ##############################################################################

#define NB_EXTENSION_TYPES 5

#define EXTENSION_TYPE_UNPROCESSED  0
#define EXTENSION_TYPE_EXTENSION 1
#define EXTENSION_TYPE_FUSION 2
#define EXTENSION_TYPE_PROCESSING 3
#define EXTENSION_TYPE_NOT_CANONICAL 4


//! represents a flag in the poset classification algorithm; related to poset_orb
it_node



class extension {

private:
   long int pt;
   int orbit_len;
   int type;
      // EXTENSION_TYPE_UNPROCESSED = unprocessed
      // EXTENSION_TYPE_EXTENSION = extension node
      // EXTENSION_TYPE_FUSION = fusion node
      // EXTENSION_TYPE_PROCESSING = currently processing
      // EXTENSION_TYPE_NOT_CANONICAL = no extension formed
```

```
      // because it is not canonical
   int data;
      // if EXTENSION_TYPE_EXTENSION: a handle to the next
      //  poset_orbit_node
      // if EXTENSION_TYPE_FUSION: a handle to a fusion element
   int data1;
      // if EXTENSION_TYPE_FUSION: node to which we are fusing
   int data2;
      // if EXTENSION_TYPE_FUSION: extension within that
      // node to which we are fusing

public:

   extension();
   ~extension();
   int get_pt();
   void set_pt(int pt);
   int get_type();
   void set_type(int type);
   int get_orbit_len();
   void set_orbit_len(int orbit_len);
   int get_data();
   void set_data(int data);
   int get_data1();
   void set_data1(int data1);
   int get_data2();
   void set_data2(int data1);
};


void print_extension_type(std::ostream &ost, int t);


// ##############################################################################
// orbit_based_testing.cpp
// ##############################################################################


#define MAX_CALLBACK 100


//! maintains a list of test functions which define a G-invariant poset

class orbit_based_testing {

public:
```

```
   poset_classification *PC;
   int max_depth;
   long int *local_S; // [max_depth]
   int nb_callback;
   int (*callback_testing[MAX_CALLBACK])(orbit_based_testing *Obt,
         long int *S, int len, void *data, int verbose_level);
   void *callback_data[MAX_CALLBACK];

   int nb_callback_no_group;
   void (*callback_testing_no_group[MAX_CALLBACK])(
         long int *S, int len,
         long int *candidates, int nb_candidates,
         long int *good_candidates, int &nb_good_candidates,
         void *data, int verbose_level);
   void *callback_data_no_group[MAX_CALLBACK];

   orbit_based_testing();
   ~orbit_based_testing();
   void init(
         poset_classification *PC,
         int max_depth,
         int verbose_level);
   void add_callback(
         int (*func)(orbit_based_testing *Obt,
               long int *S, int len,
               void *data, int verbose_level),
         void *data,
         int verbose_level);
   void add_callback_no_group(
         void (*func)(long int *S, int len,
               long int *candidates, int nb_candidates,
               long int *good_candidates,
               int &nb_good_candidates,
               void *data, int verbose_level),
         void *data,
         int verbose_level);
   void early_test_func(
      long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   void early_test_func_by_using_group(
      long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
};
```

```
// ##############################################################################
// orbit_tracer.cpp
// ##############################################################################

//! to trace a set in the poset classification algorithm


class orbit_tracer {

private:

   poset_classification *PC;

   // data for recognize:

   data_structures_groups::vector_ge *Transporter; // [PC->sz + 1]

   long int **Set; // [PC->sz + 1][PC->max_set_size]

   int *Elt1;
   int *Elt2;
   int *Elt3;


public:

   orbit_tracer();
   ~orbit_tracer();
   void init(poset_classification *PC, int verbose_level);
   data_structures_groups::vector_ge *get_transporter();
   long int *get_set_i(int i);

   void recognize_start_over(
      int size,
      int lvl, int current_node,
      int &final_node, int verbose_level);
   // Called from poset_orbit_node::recognize_recursion
   // when trace_next_point returns FALSE
   // This can happen only if f_implicit_fusion is TRUE
   void recognize_recursion(
      int size,
      int lvl, int current_node, int &final_node,
      int verbose_level);
   // this routine is called by upstep_work::recognize
   // we are dealing with a set of size len + 1.
```

```
   // but we can only trace the first len points.
   // the tracing starts at lvl = 0 with current_node = 0
   // The input set the_set[] is not modified.
   void recognize(
      long int *the_set, int size, int *transporter,
      int &final_node, int verbose_level);
   void identify(long int *data, int sz,
         int *transporter, int &orbit_at_level,
         int verbose_level);



};



// ##############################################################################
// poset_classification_control.cpp
// ##############################################################################


//! to control the behavior of the poset classification algorithm


class poset_classification_control {

public:

   int f_problem_label;
   std::string problem_label;

   int f_path;
   std::string path;

   int f_depth;
   int depth;

   int verbose_level;
   int verbose_level_group_theory;

   int f_recover;
   std::string recover_fname;

   int f_extend;
   int extend_from, extend_to;
   int extend_r, extend_m;
   std::string extend_fname;
```

```
   int f_lex;


   int f_w; // write output in level files (only last level)
   int f_W; // write output in level files (each level)
   int f_write_data_files;

   int f_T; // draw tree file (each level)
   int f_t; // draw tree file (only last level)


   int f_draw_options;
   graphics::layered_graph_draw_options *draw_options;
      // used for write_treefile in poset_classification_io


#if 0
   int f_write_tree; // create a tree

   int f_find_node_by_stabilizer_order;
   int find_node_by_stabilizer_order;


   int f_draw_poset;
   int f_draw_full_poset;


   int f_plesken;


   int f_print_data_structure;

   int f_list;
   int f_list_all;
   int f_table_of_nodes;
   int f_make_relations_with_flag_orbits;

   int f_level_summary_csv;
   int f_orbit_reps_csv;

   int f_report;
   poset_classification_report_options *report_options;

   int f_node_label_is_group_order;
   int f_node_label_is_element;
```

```
   int f_show_orbit_decomposition;
   int f_show_stab;
   int f_save_stab;
   int f_show_whole_orbits;



   int f_export_schreier_trees;
   int f_draw_schreier_trees;
   std::string schreier_tree_prefix;
         // comes after problem_label_with_path

   int f_test_multi_edge_in_decomposition_matrix;
#endif




   int f_preferred_choice;
   std::vector<std::vector<int> > preferred_choice;

   int f_clique_test;
   std::string clique_test_graph;
   graph_theory::colored_graph *clique_test_CG;




   int f_has_invariant_subset_for_root_node;
   int *invariant_subset_for_root_node;
   int invariant_subset_for_root_node_size;


   int f_do_group_extension_in_upstep;
      // is TRUE by default

   int f_allowed_to_show_group_elements;
   int downstep_orbits_print_max_orbits;
   int downstep_orbits_print_max_points_per_orbit;




   poset_classification_control();
   ~poset_classification_control();
   int read_arguments(
      int argc, std::string *argv,
```

```
      int verbose_level);
   void print();
   void prepare(
         poset_classification *PC, int verbose_level);
   void early_test_func_for_clique_search(
      long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   void init_root_node_invariant_subset(
      int *invariant_subset, int invariant_subset_size,
      int verbose_level);

};

void poset_classification_control_preferred_choice_function(
      int pt, int &pt_pref,
      groups::schreier *Sch,
      void *data, int data2,
      int verbose_level);





// ##############################################################################
// poset_classification.cpp
// ##############################################################################

//! the poset classification algorithm


class poset_classification {

private:
   int t0;

   poset_classification_control *Control;


   std::string problem_label;
      // = Control->problem_label
   std::string problem_label_with_path;
      // Control->path + Control->problem_label


   poset_with_group_action *Poset;
```

```
   int f_base_case;
   classification_base_case *Base_case;


   data_structures_groups::schreier_vector_handler
      *Schreier_vector_handler;


   int depth;


   // used as storage for the current set:
   // poset_orbit_node::store_set stores to set_S[]
   long int *set_S; // [sz]

   int sz; // = depth, the target depth
   int max_set_size; // Poset->A2->degree


   orbit_tracer *Orbit_tracer;

   poset_of_orbits *Poo;




   long int nb_times_image_of_called0;
   long int nb_times_mult_called0;
   long int nb_times_invert_called0;
   long int nb_times_retrieve_called0;
   long int nb_times_store_called0;

   double progress_last_time;
   double progress_epsilon;

public:



   // poset_classification.cpp:
   poset_of_orbits *get_Poo();
   orbit_tracer *get_Orbit_tracer();
   std::string &get_problem_label_with_path();
   std::string &get_problem_label();
   int first_node_at_level(int i);
```

```
poset_orbit_node *get_node(int node_idx);
data_structures_groups::vector_ge *get_transporter();
int *get_transporter_i(int i);
int get_sz();
int get_max_set_size();
long int *get_S();
long int *get_set_i(int i);
long int *get_set0();
long int *get_set1();
long int *get_set3();
int allowed_to_show_group_elements();
int do_group_extension_in_upstep();
poset_with_group_action *get_poset();
poset_classification_control *get_control();
actions::action *get_A();
actions::action *get_A2();
algebra::vector_space *get_VS();
data_structures_groups::schreier_vector_handler
   *get_schreier_vector_handler();
int &get_depth();
int has_base_case();
int has_invariant_subset_for_root_node();
int size_of_invariant_subset_for_root_node();
int *get_invariant_subset_for_root_node();
classification_base_case *get_Base_case();
int node_has_schreier_vector(int node_idx);
int max_number_of_orbits_to_print();
int max_number_of_points_to_print_in_orbit();
void invoke_early_test_func(
      long int *the_set, int lvl,
      long int *candidates,
      int nb_candidates,
      long int *good_candidates,
      int &nb_good_candidates,
      int verbose_level);
int nb_orbits_at_level(int level);
long int nb_flag_orbits_up_at_level(int level);
poset_orbit_node *get_node_ij(int level, int node);
int poset_structure_is_contained(long int *set1, int sz1,
   long int *set2, int sz2, int verbose_level);
data_structures_groups::orbit_transversal
   *get_orbit_transversal(
      int level, int verbose_level);
int test_if_stabilizer_is_trivial(
      int level, int orbit_at_level, int verbose_level);
data_structures_groups::set_and_stabilizer
   *get_set_and_stabilizer(int level,
```

```
   int orbit_at_level, int verbose_level);
void get_set_by_level(int level, int node, long int *set);
void get_set(int node, long int *set, int &size);
void get_set(int level, int orbit, long int *set, int &size);

int find_poset_orbit_node_for_set(
      int len, long int *set,
   int f_tolerant, int verbose_level);
int find_poset_orbit_node_for_set_basic(int from,
   int node, int len, long int *set, int f_tolerant,
   int verbose_level);
long int count_extension_nodes_at_level(int lvl);
double level_progress(int lvl);
void count_automorphism_group_orders(
      int lvl, int &nb_agos,
      ring_theory::longinteger_object *&agos,
      int *&multiplicities,
   int verbose_level);
void compute_and_print_automorphism_group_orders(int lvl,
      std::ostream &ost);
void stabilizer_order(
      int node, ring_theory::longinteger_object &go);
void orbit_length(
      int orbit_at_level, int level,
      ring_theory::longinteger_object &len);
void get_orbit_length_and_stabilizer_order(
      int node, int level,
      ring_theory::longinteger_object &stab_order,
      ring_theory::longinteger_object &len);
int orbit_length_as_int(int orbit_at_level, int level);
void recreate_schreier_vectors_up_to_level(int lvl,
   int verbose_level);
void recreate_schreier_vectors_at_level(int level,
   int verbose_level);
void find_node_by_stabilizer_order(
      int level, int order, int verbose_level);
void get_all_stabilizer_orders_at_level(int level,
      long int *&Ago, int &nb, int verbose_level);
void get_stabilizer_order(int level, int orbit_at_level,
      ring_theory::longinteger_object &go);
long int get_stabilizer_order_lint(int level,
      int orbit_at_level);
void get_stabilizer_group(
      data_structures_groups::group_container *&G,
   int level, int orbit_at_level, int verbose_level);
void get_stabilizer_generators_cleaned_up(
      groups::strong_generators *&gens,
```

```
   int level, int orbit_at_level, int verbose_level);
void get_stabilizer_generators(
      groups::strong_generators *&gens,
   int level, int orbit_at_level, int verbose_level);
void orbit_element_unrank(
      int depth, int orbit_idx,
   long int rank, long int *set, int verbose_level);
void orbit_element_rank(
      int depth, int &orbit_idx,
   long int &rank, long int *set,
   int verbose_level);
void coset_unrank(
      int depth, int orbit_idx, long int rank,
   int *Elt, int verbose_level);
long int coset_rank(
      int depth, int orbit_idx, int *Elt,
   int verbose_level);
void list_all_orbits_at_level(
      int depth,
   int f_has_print_function,
   void (*print_function)(std::ostream &ost,
         int len, long int *S, void *data),
   void *print_function_data,
   int f_show_orbit_decomposition, int f_show_stab,
   int f_save_stab, int f_show_whole_orbit);
void compute_integer_property_of_selected_list_of_orbits(
   int depth,
   int nb_orbits, int *Orbit_idx,
   int (*compute_function)(
         int len, long int *S, void *data),
   void *compute_function_data,
   int *&Data);
void list_selected_set_of_orbits_at_level(int depth,
   int nb_orbits, int *Orbit_idx,
   int f_has_print_function,
   void (*print_function)(std::ostream &ost,
         int len, long int *S, void *data),
   void *print_function_data,
   int f_show_orbit_decomposition, int f_show_stab,
   int f_save_stab, int f_show_whole_orbit);
void test_property(int depth,
   int (*test_property_function)(
         int len, long int *S, void *data),
   void *test_property_data,
   int &nb, int *&Orbit_idx);
void list_whole_orbit(int depth, int orbit_idx,
   int f_has_print_function,
```

```
   void (*print_function)(std::ostream &ost,
         int len, long int *S, void *data),
   void *print_function_data,
   int f_show_orbit_decomposition, int f_show_stab,
   int f_save_stab, int f_show_whole_orbit);
void get_whole_orbit(
   int depth, int orbit_idx,
   long int *&Orbit, int &orbit_length, int verbose_level);
void map_to_canonical_k_subset(
      long int *the_set, int set_size,
   int subset_size, int subset_rk,
   long int *reduced_set, int *transporter, int &local_idx,
   int verbose_level);
   // fills reduced_set[set_size - subset_size],
   // transporter and local_idx
   // local_idx is the index of the orbit that the
   // subset belongs to
   // (in the list of orbit of subsets of size subset_size)
void get_representative_of_subset_orbit(
      long int *set, int size,
   int local_orbit_no,
   groups::strong_generators *&Strong_gens,
   int verbose_level);
void find_interesting_k_subsets(
      long int *the_set, int n, int k,
   int *&interesting_sets, int &nb_interesting_sets,
   int &orbit_idx, int verbose_level);
void classify_k_subsets(
      long int *the_set, int n, int k,
      data_structures::tally *&C, int verbose_level);
void trace_all_k_subsets_and_compute_frequencies(
      long int *the_set,
      int n, int k, int &nCk,
      int *&isotype, int *&orbit_frequencies, int &nb_orbits,
      int verbose_level);
void trace_all_k_subsets(
      long int *the_set, int n, int k,
   int &nCk, int *&isotype, int verbose_level);
void get_orbit_representatives(int level, int &nb_orbits,
   long int *&Orbit_reps, int verbose_level);
void unrank_point(int *v, long int rk);
long int rank_point(int *v);
void unrank_basis(int *Basis, long int *S, int len);
void rank_basis(int *Basis, long int *S, int len);

// poset_classification_init.cpp:
poset_classification();
```

```
~poset_classification();
void init_internal(
   poset_classification_control *PC_control,
   poset_with_group_action *Poset,
   int sz, int verbose_level);
void initialize_and_allocate_root_node(
   poset_classification_control *PC_control,
   poset_with_group_action *Poset,
   int depth,
   int verbose_level);
void initialize_with_base_case(
   poset_classification_control *PC_control,
   poset_with_group_action *Poset,
   int depth,
   classification_base_case *Base_case,
   int verbose_level);
void init_base_case(
      classification_base_case *Base_case,
   int verbose_level);
   // Does not initialize the first starter nodes.
   // This is done in init_root_node

// poset_classification_classify.cpp:
void compute_orbits_on_subsets(
   int target_depth,
   poset_classification_control *PC_control,
   poset_with_group_action *Poset,
   int verbose_level);
int main(int t0,
   int schreier_depth,
   int f_use_invariant_subset_if_available,
   int f_debug,
   int verbose_level);
   // f_use_invariant_subset_if_available is
   // an option that affects the downstep.
   // if FALSE, the orbits of the stabilizer
   // on all points are computed.
   // if TRUE, the orbits of the stabilizer
   // on the set of points that were
   // possible in the previous level are computed only
   // (using Schreier.orbits_on_invariant_subset_fast).
   // The set of possible points is stored
   // inside the schreier vector data structure (sv).
int compute_orbits(
      int from_level, int to_level,
      int schreier_depth,
      int f_use_invariant_subset_if_available,
```

```
      int verbose_level);
      // returns the last level that has at least one orbit
//void post_processing(int actual_size, int verbose_level);
void recognize(
      std::string &set_to_recognize,
      int h, int nb_to_recognize, int verbose_level);
void extend_level(
      int size,
   int f_create_schreier_vector,
   int f_use_invariant_subset_if_available,
   int f_debug,
   int f_write_candidate_file,
   int verbose_level);
   // calls compute_flag_orbits, upstep
void compute_flag_orbits(
      int size,
   int f_create_schreier_vector,
   int f_use_invariant_subset_if_available,
   int verbose_level);
   // calls root[prev].downstep_subspace_action
   // or root[prev].downstep
void upstep(
      int size, int f_debug,
   int verbose_level);
   // calls extend_node
void extend_node(
      int size,
   int prev, int &cur,
   int f_debug,
   int f_indicate_not_canonicals,
   int verbose_level);
   // called by poset_classification::upstep
   // Uses an upstep_work structure to handle the work.
   // Calls upstep_work::handle_extension



// poset_classification_combinatorics.cpp
void compute_Kramer_Mesner_matrix(int t, int k,
      int verbose_level);
void Plesken_matrix_up(int depth,
   int *&P, int &N, int verbose_level);
void Plesken_matrix_down(int depth,
   int *&P, int &N, int verbose_level);
void Plesken_submatrix_up(int i, int j,
   int *&Pij, int &N1, int &N2, int verbose_level);
void Plesken_submatrix_down(int i, int j,
```

```
   int *&Pij, int &N1, int &N2, int verbose_level);
int count_incidences_up(int lvl1, int po1,
   int lvl2, int po2, int verbose_level);
int count_incidences_down(int lvl1,
   int po1, int lvl2, int po2, int verbose_level);
void Asup_to_Ainf(int t, int k, int *M_sup,
   int *M_inf, int verbose_level);
void test_for_multi_edge_in_classification_graph(
   int depth, int verbose_level);
void Kramer_Mesner_matrix_neighboring(
      int level, long int *&M,
      int &nb_rows, int &nb_cols, int verbose_level);
void Mtk_via_Mtr_Mrk(
      int t, int r, int k,
      long int *Mtr, long int *Mrk, long int *&Mtk,
      int nb_r1, int nb_c1, int nb_r2, int nb_c2,
      int &nb_r3, int &nb_c3,
      int verbose_level);
// Computes $M_{tk}$ via a recursion formula:
// $M_{tk} = {{k - t} \choose {k - r}} \cdot M_{t,r} \cdot M_{r,k}$.
void Mtk_from_MM(
      long int **pM,
   int *Nb_rows, int *Nb_cols,
   int t, int k,
   long int *&Mtk, int &nb_r, int &nb_c,
   int verbose_level);


// poset_classification_draw.cpp:
void draw_poset_fname_base_aux_poset(std::string &fname, int depth);
void draw_poset_fname_base_poset_lvl(std::string &fname, int depth);
void draw_poset_fname_base_tree_lvl(std::string &fname, int depth);
void draw_poset_fname_base_poset_detailed_lvl(std::string &fname, int depth);
void draw_poset_fname_aux_poset(std::string &fname, int depth);
void draw_poset_fname_poset(std::string &fname, int depth);
void draw_poset_fname_tree(std::string &fname, int depth);
void draw_poset_fname_poset_detailed(std::string &fname, int depth);
void write_treefile(std::string &fname_base,
   int lvl,
   graphics::layered_graph_draw_options *draw_options,
   int verbose_level);
int write_treefile(std::string &fname_base, int lvl,
   int verbose_level);
void draw_tree(
      std::string &fname_base, int lvl,
      graphics::tree_draw_options *Tree_draw_options,
      graphics::layered_graph_draw_options *Draw_options,
```

```
   int xmax, int ymax, int rad, int f_embedded,
   int f_sideways, int verbose_level);
void draw_tree_low_level(
      std::string &fname,
      graphics::tree_draw_options *Tree_draw_options,
      graphics::layered_graph_draw_options *Draw_options,
      int nb_nodes,
   int *coord_xyw, int *perm, int *perm_inv,
   int f_draw_points, int f_draw_extension_points,
   int f_draw_aut_group_order,
   int xmax, int ymax, int rad, int f_embedded,
   int f_sideways,
   int verbose_level);
void draw_tree_low_level1(
      graphics::mp_graphics &G, int nb_nodes,
   int *coords, int *perm, int *perm_inv,
   int f_draw_points, int f_draw_extension_points,
   int f_draw_aut_group_order,
   int radius, int verbose_level);
void draw_poset_full(
      std::string &fname_base,
      int depth, int data,
      graphics::layered_graph_draw_options *LG_Draw_options,
      double x_stretch,
      int verbose_level);
void draw_poset(
      std::string &fname_base,
      int depth, int data,
      graphics::layered_graph_draw_options *LG_Draw_options,
      int verbose_level);
void draw_level_graph(
      std::string &fname_base,
      int depth, int data, int level,
      graphics::layered_graph_draw_options *LG_Draw_options,
      int verbose_level);
void make_flag_orbits_on_relations(
      int depth,
      std::string &fname_prefix, int verbose_level);
void make_full_poset_graph(
      int depth,
      graph_theory::layered_graph *&LG,
   int data1, double x_stretch,
   int verbose_level);
   // Draws the full poset: each element of each orbit is drawn.
   // The orbits are indicated by grouping the elements closer together.
   // Uses int_vec_sort_and_test_if_contained to test containment relation.
   // This is only good for actions on sets, not for actions on subspaces
```

```
void make_auxiliary_graph(int depth,
      graph_theory::layered_graph *&LG, int data1,
   int verbose_level);
   // makes a graph of the poset of orbits with 2 * depth + 1 layers.
   // The middle layers represent the flag orbits.
void make_graph(
      int depth, graph_theory::layered_graph *&LG,
   int data1, int f_tree, int verbose_level);
   // makes a graph  of the poset of orbits with depth + 1 layers.
void make_level_graph(
      int depth,
      graph_theory::layered_graph *&LG,
   int data1, int level, int verbose_level);
   // makes a graph with 4 levels showing the relation between
   // orbits at level ´level´ and orbits at level ´level´ + 1
void make_poset_graph_detailed(
      graph_theory::layered_graph *&LG,
   int data1, int max_depth, int verbose_level);
   // creates the poset graph, with two middle layers at each level.
   // In total, the graph that is created will have 3 * depth + 1 layers.
void print_data_structure_tex(
      int depth, int verbose_level);


// in poset_classification_io.cpp:
void print_set_verbose(int node);
void print_set_verbose(int level, int orbit);
void print_set(int node);
void print_set(int level, int orbit);
void print_progress_by_extension(
      int size, int cur,
   int prev, int cur_ex, int nb_ext_cur, int nb_fuse_cur);
void print_progress(int size, int cur, int prev,
   int nb_ext_cur, int nb_fuse_cur);
void print_progress(double progress);
void print_progress_by_level(int lvl);
void print_orbit_numbers(int depth);
void print();
void print_statistic_on_callbacks_naked();
void print_statistic_on_callbacks();
void prepare_fname_data_file(std::string &fname,
      std::string &fname_base, int depth_completed);
void print_representatives_at_level(int lvl);
void print_lex_rank(long int *set, int sz);
void print_problem_label();
void print_level_info(int prev_level, int prev);
void print_level_extension_info(int prev_level,
```

```
   int prev, int cur_extension);
void print_level_extension_coset_info(int prev_level,
   int prev, int cur_extension, int coset, int nb_cosets);
void print_node(int node);
void print_extensions_at_level(
      std::ostream &ost, int lvl);
void print_fusion_nodes(int depth);
void read_data_file(
      int &depth_completed,
   std::string &fname, int verbose_level);
void write_data_file(int depth_completed,
      std::string &fname_base, int verbose_level);
void write_file(
      std::ofstream &fp, int depth_completed,
   int verbose_level);
void read_file(
      std::ifstream &fp, int &depth_completed,
   int verbose_level);
void housekeeping(int i, int f_write_files, int t0,
   int verbose_level);
void housekeeping_no_data_file(
      int i, int t0, int verbose_level);
void create_fname_sv_level_file_binary(
      std::string &fname,
      std::string &fname_base, int level);
int test_sv_level_file_binary(
      int level, std::string &fname_base);
void read_sv_level_file_binary(
      int level, std::string &fname_base,
   int f_split, int split_mod, int split_case,
   int f_recreate_extensions, int f_dont_keep_sv,
   int verbose_level);
void write_sv_level_file_binary(
      int level, std::string &fname_base,
   int f_split, int split_mod, int split_case,
   int verbose_level);
void read_level_file_binary(
      int level, std::string &fname_base,
   int verbose_level);
void write_level_file_binary(
      int level, std::string &fname_base,
   int verbose_level);
void recover(std::string &recover_fname,
   int &depth_completed, int verbose_level);
void make_fname_lvl_file_candidates(
      std::string &fname,
      std::string &fname_base, int lvl);
```

```
void make_fname_lvl_file(
      std::string &fname,
      std::string &fname_base, int lvl);
void make_fname_lvl_reps_file(
      std::string &fname,
      std::string &fname_base, int lvl);
void log_current_node(
      std::ostream &f, int size);

void make_spreadsheet_of_orbit_reps(
      data_structures::spreadsheet *&Sp,
   int max_depth);
void make_spreadsheet_of_level_info(
      data_structures::spreadsheet *&Sp,
   int max_depth, int verbose_level);

void create_schreier_tree_fname_mask_base(
      std::string &fname_mask);
void create_schreier_tree_fname_mask_base_tex(
      std::string &fname_mask);
void create_shallow_schreier_tree_fname_mask_base(
      std::string &fname_mask);
void create_shallow_schreier_tree_fname_mask(
      std::string &fname, int node);
void make_fname_candidates_file_default(
      char *fname2000, int level);
void wedge_product_export_magma(
      int n, int q, int vector_space_dimension,
      int level, int verbose_level);
void write_reps_csv(int lvl, int verbose_level);

// poset_classification_export_source_code.cpp:
void generate_source_code(int level, int verbose_level);
void generate_history(int level, int verbose_level);



// in poset_classification_report.cpp:
void report(
      poset_classification_report_options *Opt,
      int verbose_level);
void report2(
      std::ostream &ost,
      poset_classification_report_options *Opt,
      int verbose_level);
void report_orbits_in_detail(
      std::ostream &ost,
```

```
      poset_classification_report_options *Opt,
      int verbose_level);
void report_number_of_orbits_at_level(
      std::ostream &ost,
      poset_classification_report_options *Opt,
      int verbose_level);
void report_orbits_summary(
      std::ostream &ost,
      poset_classification_report_options *Opt,
      int verbose_level);
void report_poset_of_orbits(
      std::ostream &ost, int verbose_level);
void report_orbit(
      int level, int orbit_at_level,
      poset_classification_report_options *Opt,
      std::ostream &ost, int verbose_level);

// poset_classification_trace.cpp:
int find_isomorphism(
      long int *set1, long int *set2, int sz,
   int *transporter, int &orbit_idx, int verbose_level);
void identify_and_get_stabilizer(
      long int *set, int sz, int *transporter,
      int &orbit_at_level,
      data_structures_groups::set_and_stabilizer
         *&Set_and_stab_original,
      data_structures_groups::set_and_stabilizer
         *&Set_and_stab_canonical,
      int verbose_level);
void test_identify(
      int level, int nb_times, int verbose_level);
void poset_classification_apply_isomorphism_no_transporter(
   int cur_level, int size, int cur_node, int cur_ex,
   long int *set_in, long int *set_out,
   int verbose_level);
int poset_classification_apply_isomorphism(
      int level, int size,
   int current_node, int current_extension,
   long int *set_in, long int *set_out, long int *set_tmp,
   int *transporter_in, int *transporter_out,
   int f_tolerant,
   int verbose_level);
int trace_set_recursion(
      int cur_level, int cur_node,
   int size, int level,
   long int *canonical_set,
   long int *tmp_set1, long int *tmp_set2,
```

```
      int *Elt_transporter, int *tmp_Elt1, int *tmp_Elt2,
      int f_tolerant,
      int verbose_level);
      // called by poset_classification::trace_set
      // returns the node in the poset_classification that corresponds
      // to the canonical_set
      // or -1 if f_tolerant and the node could not be found
   int trace_set(
         long int *set, int size, int level,
      long int *canonical_set, int *Elt_transporter,
      int verbose_level);
      // returns the case number of the canonical set
   long int find_node_for_subspace_by_rank(
         long int *set, int len,
      int verbose_level);

};


const char *trace_result_as_text(trace_result r);
int trace_result_is_no_result(trace_result r);



// ##############################################################################
// poset_classification_report_options.cpp
// ##############################################################################


//! to control the behavior of the poset classification report function


class poset_classification_report_options {

public:

   int f_select_orbits_by_level;
   int select_orbits_by_level_level;

   int f_select_orbits_by_stabilizer_order;
   int select_orbits_by_stabilizer_order_so;

   int f_select_orbits_by_stabilizer_order_multiple_of;
   int select_orbits_by_stabilizer_order_so_multiple_of;

   int f_include_projective_stabilizer;
```

```
   int f_draw_poset;

   int f_fname;
   std::string fname;

   poset_classification_report_options();
   ~poset_classification_report_options();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();
   int is_selected_by_group_order(long int so);

};




// ##############################################################################
// poset_of_orbits.cpp
// ##############################################################################

//! the data structure for the poset of orbits in the poset classification algori
thm


class poset_of_orbits {

private:

   poset_classification *PC;

   int sz;
   int max_set_size;
   long int t0;

   long int nb_poset_orbit_nodes_used;
   long int nb_poset_orbit_nodes_allocated;
   long int poset_orbit_nodes_increment;
   long int poset_orbit_nodes_increment_last;

   poset_orbit_node *root;

   long int *first_poset_orbit_node_at_level;

   long int *nb_extension_nodes_at_level_total;
   long int *nb_extension_nodes_at_level;
```

```
   long int *nb_fusion_nodes_at_level;
   long int *nb_unprocessed_nodes_at_level;

public:
   long int *set0; // [max_set_size] temporary storage
   long int *set1; // [max_set_size] temporary storage
   long int *set3; // [max_set_size] temporary storage


   poset_of_orbits();
   ~poset_of_orbits();
   void init(poset_classification *PC,
         int nb_poset_orbit_nodes,
         int sz, int max_set_size, long int t0,
         int verbose_level);
   void init_poset_orbit_node(
         int nb_poset_orbit_nodes,
         int verbose_level);
   void reallocate();
   void reallocate_to(long int new_number_of_nodes,
         int verbose_level);
   long int get_nb_poset_orbit_nodes_allocated();
   long int get_nb_extension_nodes_at_level_total(int level);
   void set_nb_poset_orbit_nodes_used(int value);
   int first_node_at_level(int i);
   void set_first_node_at_level(int i, int value);
   poset_orbit_node *get_node(int node_idx);
   long int *get_set0();
   long int *get_set1();
   long int *get_set3();
   int nb_orbits_at_level(int level);
   long int nb_flag_orbits_up_at_level(int level);
   poset_orbit_node *get_node_ij(int level, int node);
   int node_get_nb_of_extensions(int node);
   void get_set(
         int node, long int *set, int &size);
   void get_set(
         int level, int orbit, long int *set, int &size);
   int find_extension_from_point(
         int node_idx,
         long int pt, int verbose_level);
   long int count_extension_nodes_at_level(int lvl);
   double level_progress(int lvl);
   void change_extension_type(int level,
         int node, int cur_ext, int type, int verbose_level);
   void get_table_of_nodes(long int *&Table,
         int &nb_rows, int &nb_cols, int verbose_level);
```

```
int count_live_points(
      int level,
      int node_local, int verbose_level);
void print_progress_by_level(int lvl);
void print_tree();
void init_root_node_from_base_case(int verbose_level);
void init_root_node(int verbose_level);
void make_tabe_of_nodes(int verbose_level);
void poset_orbit_node_depth_breadth_perm_and_inverse(
   int max_depth,
   int *&perm, int *&perm_inv, int verbose_level);
void read_memory_object(
      int &depth_completed,
      orbiter_kernel_system::memory_object *m,
      int &nb_group_elements,
      int verbose_level);
void write_memory_object(
      int depth_completed,
      orbiter_kernel_system::memory_object *m,
      int &nb_group_elements,
      int verbose_level);
long int calc_size_on_file(int depth_completed,
      int verbose_level);
void read_sv_level_file_binary2(
   int level, std::ifstream &fp,
   int f_split, int split_mod, int split_case,
   int f_recreate_extensions, int f_dont_keep_sv,
   int verbose_level);
void write_sv_level_file_binary2(
   int level, std::ofstream &fp,
   int f_split, int split_mod, int split_case,
   int verbose_level);
void read_level_file_binary2(
   int level, std::ifstream &fp,
   int &nb_group_elements, int verbose_level);
void write_level_file_binary2(
   int level, std::ofstream &fp,
   int &nb_group_elements, int verbose_level);
void write_candidates_binary_using_sv(
      const char *fname_base,
      int lvl, int t0, int verbose_level);
void read_level_file(int level,
      std::string &fname, int verbose_level);
void write_lvl_file_with_candidates(
      std::string &fname_base, int lvl, int t0,
      int verbose_level);
void get_orbit_reps_at_level(
```

```
        int lvl, long int *&Data,
        int &nb_reps, int verbose_level);
   void write_orbit_reps_at_level(
        std::string &fname_base,
        int lvl,
        int verbose_level);
   void write_lvl_file(
        std::string &fname_base,
        int lvl, int t0, int f_with_stabilizer_generators,
        int f_long_version,
        int verbose_level);
   void write_lvl(
        std::ostream &f, int lvl, int t0,
        int f_with_stabilizer_generators, int f_long_version,
        int verbose_level);
   void log_nodes_for_treefile(
        int cur, int depth,
        std::ostream &f, int f_recurse, int verbose_level);
   void save_representatives_at_level_to_csv(
        std::string &fname,
        int lvl, int verbose_level);

};



// ##############################################################################
// poset_description.cpp
// ##############################################################################

//! description of a poset from the command line


class poset_description {
public:
   int f_subset_lattice;

   int f_subspace_lattice;
   int dimension;
   int q;


   int f_independence_condition;
   int independence_condition_value;


   poset_description();
```

```
   ~poset_description();
   void read_arguments_from_string(
         const char *str, int verbose_level);
   int read_arguments(int argc, const char **argv,
      int verbose_level);
};




// ##############################################################################
// poset_orbit_node.cpp, poset_orbit_node_io.cpp, poset_orbit_node_upstep.cpp,
// poset_orbit_node_upstep_subspace_action.cpp,
// poset_orbit_node_downstep.cpp, poset_orbit_node_downstep_subspace_action.cpp:
// ##############################################################################


//! to represent one poset orbit; related to the class poset_classification


class poset_orbit_node {

private:
   int node;
   int prev;

   long int pt;
   int nb_strong_generators;
   int first_strong_generator_handle;
      // only if there is a generator
   int *tl;
      // only if the group is not trivial

   int nb_extensions;
   extension *E;

   data_structures_groups::schreier_vector *Schreier_vector;

   actions::action *A_on_upset;
   // only used for actions on subspace lattices
   // it records the action on the factor space
   // modulo the current subspace
   // used in poset_orbit_node_downstep_subspace_action.cpp
   // and poset_orbit_node_upstep_subspace_action.cpp

public:
```

```
// poset_orbit_node.cpp:
poset_orbit_node();
~poset_orbit_node();
void null();
void freeself();
void init_root_node(
      poset_classification *gen, int verbose_level);
   // copies gen->SG0 and gen->tl into the poset_orbit_node
   // structure using store_strong_generators
void init_node(
      int node, int prev, long int pt, int verbose_level);
int get_node();
void set_node(int node);
void delete_Schreier_vector();
void allocate_E(int nb_extensions, int verbose_level);
int get_level(poset_classification *gen);
int get_node_in_level(poset_classification *gen);
void get_strong_generators_handle(
      std::vector<int> &gen_hdl, int verbose_level);
void get_tl(
      std::vector<int> &tl,
      poset_classification *PC, int verbose_level);
int get_tl(int i);
int has_Schreier_vector();
data_structures_groups::schreier_vector
   *get_Schreier_vector();
int get_nb_strong_generators();
int *live_points();
int get_nb_of_live_points();
int get_nb_of_orbits_under_stabilizer();
int get_nb_of_extensions();
extension *get_E(int idx);
long int get_pt();
void set_pt(long int pt);
int get_prev();
void set_prev(int prev);
void poset_orbit_node_depth_breadth_perm_and_inverse(
   poset_classification *gen,
   int max_depth,
   int &idx, int hdl, int cur_depth, int *perm, int *perm_inv);
int find_extension_from_point(
      poset_classification *gen, long int pt,
   int verbose_level);
void print_extensions(
      std::ostream &ost);
void log_current_node_without_group(
```

```
      poset_classification *gen,
   int s, std::ostream &f, int verbose_level);
void log_current_node(
      poset_classification *gen, int s,
      std::ostream &f, int f_with_stabilizer_poset_classifications,
   int verbose_level);
void log_current_node_after_applying_group_element(
   poset_classification *gen, int s,
   std::ostream &f, int hdl,
   int verbose_level);
void log_current_node_with_candidates(
      poset_classification *gen,
   int lvl, std::ostream &f, int verbose_level);
int depth_of_node(poset_classification *gen);
void store_set(poset_classification *gen, int i);
   // stores a set of size i + 1 to gen->S
void store_set_with_verbose_level(
      poset_classification *gen, int i,
   int verbose_level);
// stores a set of size i + 1 to gen->S[]
void store_set_to(
      poset_classification *gen, int i, long int *to);
void store_set_to(
      poset_classification *gen, long int *to);
int check_node_and_set_consistency(
      poset_classification *gen,
   int i, long int *set);
void print_set_verbose(
      poset_classification *gen);
void print_set(
      poset_classification *gen);
void print_node(
      poset_classification *gen);
void print_extensions(
      poset_classification *gen);
void reconstruct_extensions_from_sv(
      poset_classification *gen,
   int verbose_level);
int nb_extension_points();
   // sums up the lengths of orbits in all extensions

// in poset_orbit_node_group_theory.cpp:
void store_strong_generators(
      poset_classification *gen,
      groups::strong_generators *Strong_gens);
void get_stabilizer_order(
      poset_classification *gen,
```

```
      ring_theory::longinteger_object &go);
long int get_stabilizer_order_lint(poset_classification *PC);
void get_stabilizer(poset_classification *PC,
      data_structures_groups::group_container &G,
      ring_theory::longinteger_object &go_G,
   int verbose_level);
int test_if_stabilizer_is_trivial();
void get_stabilizer_generators(poset_classification *PC,
      groups::strong_generators *&Strong_gens,
   int verbose_level);
void init_extension_node_prepare_G(
   poset_classification *gen,
   int prev, int prev_ex, int size,
   data_structures_groups::group_container &G,
   ring_theory::longinteger_object &go_G,
   int verbose_level);
   // sets up the group G using the strong
   // poset_classifications that are stored
void init_extension_node_prepare_H(
   poset_classification *gen,
   int prev, int prev_ex, int size,
   data_structures_groups::group_container &G,
   ring_theory::longinteger_object &go_G,
   data_structures_groups::group_container &H,
   ring_theory::longinteger_object &go_H,
   long int pt, int pt_orbit_len,
   int verbose_level);
   // sets up the group H which is the stabilizer
   // of the point pt in G
void compute_point_stabilizer_in_subspace_setting(
   poset_classification *gen,
   int prev, int prev_ex, int size,
   data_structures_groups::group_container &G,
   ring_theory::longinteger_object &go_G,
   data_structures_groups::group_container &H,
   ring_theory::longinteger_object &go_H,
   long int pt, int pt_orbit_len,
   int verbose_level);
void compute_point_stabilizer_in_standard_setting(
   poset_classification *gen,
   int prev, int prev_ex, int size,
   data_structures_groups::group_container &G,
   ring_theory::longinteger_object &go_G,
   data_structures_groups::group_container &H,
   int pt, int pt_orbit_len,
   int verbose_level);
void create_schreier_vector_wrapper(
```

```
      poset_classification *gen,
      int f_create_schreier_vector,
      groups::schreier *Schreier, int verbose_level);
      // called from downstep_orbit_test_and_schreier_vector
      // calls Schreier.get_schreier_vector
   void create_schreier_vector_wrapper_subspace_action(
      poset_classification *gen,
      int f_create_schreier_vector,
      groups::schreier &Schreier,
      actions::action *A_factor_space,
      induced_actions::action_on_factor_space *AF,
      int verbose_level);


   // in poset_orbit_node_io.cpp:
   void read_memory_object(
      poset_classification *PC,
      actions::action *A,
      orbiter_kernel_system::memory_object *m,
      int &nb_group_elements,
      int *Elt_tmp,
      int verbose_level);
   void write_memory_object(
      poset_classification *PC,
      actions::action *A,
      orbiter_kernel_system::memory_object *m,
      int &nb_group_elements,
      int *Elt_tmp,
      int verbose_level);
   long int calc_size_on_file(
         actions::action *A, int verbose_level);
   void sv_read_file(
      poset_classification *PC,
      std::ifstream &fp, int verbose_level);
   void sv_write_file(
      poset_classification *PC,
      std::ofstream &fp, int verbose_level);
   void read_file(
         actions::action *A,
         std::ifstream &fp, int &nb_group_elements,
      int verbose_level);
   void write_file(
         actions::action *A,
         std::ofstream &fp, int &nb_group_elements,
      int verbose_level);
#if 0
   void save_schreier_forest(
```

```
      poset_classification *PC,
      groups::schreier *Schreier,
      int verbose_level);
   void save_shallow_schreier_forest(
      poset_classification *PC,
      int verbose_level);
   void draw_schreier_forest(
      poset_classification *PC,
      groups::schreier *Schreier,
      int f_using_invariant_subset, actions::action *AR,
      int verbose_level);
#endif

   // poset_orbit_node_upstep.cpp:
   int apply_isomorphism(
         poset_classification *gen,
      int lvl, int current_node,
      int current_extension, int len, int f_tolerant,
      int *Elt_tmp1, int *Elt_tmp2,
      int verbose_level);
      // returns next_node
   void install_fusion_node(
         poset_classification *gen,
      int lvl, int current_node,
      int my_node, int my_extension, int my_coset,
      long int pt0, int current_extension,
      int f_debug, int f_implicit_fusion,
      int *Elt_tmp,
      int verbose_level);
      // Called from poset_orbit_node::handle_last_level
   int trace_next_point_wrapper(
         poset_classification *gen, int lvl,
      int current_node,
      int len, int f_implicit_fusion,
      int *cosetrep,
      int &f_failure_to_find_point,
      int verbose_level);
      // Called from upstep_work::recognize_recursion
      // applies the permutation which maps the point with index lvl
      // (i.e. the lvl+1-st point) to its orbit representative.
      // also maps all the other points under that permutation.
      // we are dealing with a set of size len + 1
      // returns FALSE if we are using implicit fusion nodes
      // and the set becomes lexicographically
      // less than before, in which case trace has to be restarted.
   int trace_next_point_in_place(
         poset_classification *gen,
```

```
   int lvl, int current_node, int size,
   long int *cur_set, long int *tmp_set,
   int *cur_transporter, int *tmp_transporter,
   int *cosetrep,
   int f_implicit_fusion, int &f_failure_to_find_point,
   int verbose_level);
   // called by poset_classification::trace_set_recursion
void trace_starter(
      poset_classification *gen, int size,
   long int *cur_set, long int *next_set,
   int *cur_transporter, int *next_transporter,
   int verbose_level);
int trace_next_point(
      poset_classification *gen,
   int lvl, int current_node, int size,
   long int *cur_set, long int *next_set,
   int *cur_transporter, int *next_transporter,
   int *cosetrep,
   int f_implicit_fusion, int &f_failure_to_find_point,
   int verbose_level);
   // Called by poset_orbit_node::trace_next_point_wrapper
   // and by poset_orbit_node::trace_next_point_in_place
   // returns FALSE only if f_implicit_fusion is TRUE and
   // the set becomes lexicographically less
int orbit_representative_and_coset_rep_inv(
      poset_classification *gen,
   int lvl, long int pt_to_trace, long int &pt0, int *cosetrep,
   int verbose_level);
   // called by poset_orbit_node::trace_next_point
   // FALSE means the point to trace was not found.
   // This can happen if nodes were eliminated due to clique_test

// poset_orbit_node_upstep_subspace_action.cpp:
void orbit_representative_and_coset_rep_inv_subspace_action(
   poset_classification *gen,
   int lvl, long int pt_to_trace, long int &pt0, int *cosetrep,
   int verbose_level);
   // called by poset_orbit_node::trace_next_point


// poset_orbit_node_downstep.cpp
// top level functions:
void compute_flag_orbits(
   poset_classification *gen,
   int lvl,
   int f_create_schreier_vector,
   int f_use_invariant_subset_if_available,
```

```
   int f_implicit_fusion,
   int verbose_level);
   // Called from poset_classification::compute_flag_orbits
   // if we are acting on sets
   // (i.e., not on subspaces).
   // Calls downstep_orbits,
   // downstep_orbit
void compute_schreier_vector(
      poset_classification *gen,
   int lvl, int verbose_level);
   // called from poset_classification::recreate_schreier_vectors_at_level
   // and from poset_classification::count_live_points
   // calls downstep_apply_early_test
   // and check_orbits
   // and Schreier.get_schreier_vector
void get_candidates(
   poset_classification *gen,
   int lvl,
   long int *&candidates, int &nb_candidates,
   int verbose_level);

   // 1st level under downstep:
void schreier_forest(
   poset_classification *gen,
   groups::schreier &Schreier, actions::action *&AR,
   int lvl,
   int f_use_invariant_subset_if_available,
   int &f_using_invariant_subset,
   int verbose_level);
   // calls downstep_get_invariant_subset,
   // downstep_apply_early_test,
   // and AR.induced_action_by_restriction
   // if f_use_invariant_subset_if_available and
   // f_using_invariant_subset
   //
   // Sets up the schreier data structure Schreier
   // If f_using_invariant_subset, we will use the
   // restricted action AR, otherwise the action gen->A2
   // In this action, the orbits are computed using
   // Schreier.compute_all_point_orbits
   // and possibly printed using downstep_orbits_print
void downstep_orbit_test_and_schreier_vector(
   poset_classification *gen,
   groups::schreier *Schreier, actions::action *AR,
   int lvl,
   int f_use_invariant_subset_if_available,
   int f_using_invariant_subset,
```

```
   int f_create_schreier_vector,
   int &nb_good_orbits, int &nb_points,
   int verbose_level);
   // called from downstep once downstep_orbits is completed
   // Calls check_orbits_wrapper and
   // create_schreier_vector_wrapper
   // The order in which these two functions are called matters.
void downstep_implicit_fusion(
   poset_classification *gen,
   groups::schreier &Schreier, actions::action *AR,
   int f_using_invariant_subset,
   int lvl,
   int f_implicit_fusion,
   int good_orbits1, int nb_points1,
   int verbose_level);
   // called from downstep,
   // once downstep_orbit_test_and_schreier_vector is done
   // calls test_orbits_for_implicit_fusion
void find_extensions(
      poset_classification *gen,
      groups::schreier &O,
      actions::action *AR, int f_using_invariant_subset,
   int lvl,
   int verbose_level);
   // called by downstep
   // prepares all extension nodes and marks them as unprocessed.
   // we are at depth lvl, i.e., currently,
   // we have a set of size lvl.


   // second level under downstep:
int downstep_get_invariant_subset(
   poset_classification *gen,
   int lvl,
   int &n, long int *&subset,
   int verbose_level);
   // called from downstep_orbits
   // Gets the live points at the present node.
void downstep_apply_early_test(
   poset_classification *gen,
   int lvl,
   int n, long int *subset,
   long int *candidates, int &nb_candidates,
   int verbose_level);
   // called from downstep_orbits, compute_schreier_vector
   // calls the callback early test function if available
   // and calls test_point_using_check_functions otherwise
```

```
void check_orbits_wrapper(
      poset_classification *gen,
      groups::schreier *Schreier, actions::action *AR,
   int lvl,
   int &nb_good_orbits1, int &nb_points1,
   //int f_use_incremental_test_func_if_available,
   int verbose_level);
   // called from downstep_orbit_test_and_schreier_vector
   // This function and create_schreier_vector_wrapper
   // are used in pairs.
   // Except, the order in which the function is used matters.
   // Calls check_orbits

void test_orbits_for_implicit_fusion(
      poset_classification *gen,
      groups::schreier &Schreier, actions::action *AR,
   int f_using_invariant_subset,
   int lvl, int verbose_level);
   // called from downstep_implicit_fusion
   // eliminates implicit fusion orbits from the
   // Schreier data structure,
void check_orbits(
      poset_classification *gen,
      groups::schreier *Schreier, actions::action *AR,
   //int f_using_invariant_subset,
   int lvl,
   //int f_use_incremental_test_func_if_available,
   int verbose_level);
   // called from compute_schreier_vector
   // and check_orbits_wrapper (which is called from
   // downstep_orbit_test_and_schreier_vector)
   // calls test_point_using_check_functions
   // eliminates bad orbits from the Schreier data structure,
   // does not eliminate implicit fusion orbits
int test_point_using_check_functions(
      poset_classification *gen,
   int lvl, int rep, int *the_set,
   int verbose_level);
   // called by check_orbits and downstep_apply_early_test
   // Calls gen->check_the_set_incrementally
   // (if gen->f_candidate_incremental_check_func).
   // Otherwise, calls gen->check_the_set
   // (if gen->f_candidate_check_func).
   // Otherwise accepts any point.
void relabel_schreier_vector(
      actions::action *AR, int verbose_level);
```

```
   // called from compute_schreier_vector,
   // downstep_orbit_test_and_schreier_vector
void downstep_orbits_print(
      poset_classification *gen,
      groups::schreier *Schreier, actions::action *AR,
   int lvl,
   int f_print_orbits,
   int max_orbits, int max_points_per_orbit);


// poset_orbit_node_downstep_subspace_action.cpp
void compute_flag_orbits_subspace_action(
   poset_classification *gen,
   int lvl,
   int f_create_schreier_vector,
   int f_use_invariant_subset_if_available,
   int f_implicit_fusion,
   int verbose_level);
   // called from poset_classification::downstep
   // creates action *A_factor_space
   // and action_on_factor_space *AF
   // and disposes them at the end.
void setup_factor_space_action_light(
   poset_classification *gen,
   induced_actions::action_on_factor_space &AF,
   int lvl, int verbose_level);
void setup_factor_space_action_with_early_test(
   poset_classification *gen,
   induced_actions::action_on_factor_space &AF,
   actions::action &A_factor_space,
   int lvl, int verbose_level);
void setup_factor_space_action(
   poset_classification *gen,
   induced_actions::action_on_factor_space &AF,
   actions::action &A_factor_space,
   int lvl, int f_compute_tables,
   int verbose_level);
void downstep_subspace_action_print_orbits(
   poset_classification *gen,
   groups::schreier &Schreier,
   int lvl,
   int f_print_orbits,
   int verbose_level);
void downstep_orbits_subspace_action(
   poset_classification *gen, groups::schreier &Schreier,
   int lvl,
   int f_use_invariant_subset_if_available,
```

```
      int &f_using_invariant_subset,
      int verbose_level);
   void find_extensions_subspace_action(
      poset_classification *gen, groups::schreier &O,
      actions::action *A_factor_space,
      induced_actions::action_on_factor_space *AF,
      int lvl, int f_implicit_fusion, int verbose_level);
};


// ##############################################################################
// poset_with_group_action.cpp
// ##############################################################################

//! a poset with a group action on it


class poset_with_group_action {
public:
   poset_description *description;

   int f_subset_lattice;
   int n;

   int f_subspace_lattice;
   algebra::vector_space *VS;

   actions::action *A; // the action in which the group is given
   actions::action *A2; // the action in which we do the search

   groups::strong_generators *Strong_gens;
   ring_theory::longinteger_object go;

   int f_has_orbit_based_testing;
   orbit_based_testing *Orbit_based_testing;

   int f_print_function;
   void (*print_function)(std::ostream &ost,
         int len, long int *S, void *data);
   void *print_function_data;

   poset_with_group_action();
   ~poset_with_group_action();
   void init_subset_lattice(
         actions::action *A, actions::action *A2,
         groups::strong_generators *Strong_gens,
         int verbose_level);
```

```
   void init_subspace_lattice(
         actions::action *A, actions::action *A2,
         groups::strong_generators *Strong_gens,
         algebra::vector_space *VS,
         int verbose_level);
   void init(
         poset_description *description,
         actions::action *A, // the action in which the group is given
         actions::action *A2, // the action in which we do the search
         groups::strong_generators *Strong_gens,
      int verbose_level);
   void add_independence_condition(
         int independence_value,
         int verbose_level);
   void add_testing(
         int (*func)(orbit_based_testing *Obt,
               long int *S, int len,
               void *data, int verbose_level),
         void *data,
         int verbose_level);
   void add_testing_without_group(
         void (*func)(long int *S, int len,
               long int *candidates, int nb_candidates,
               long int *good_candidates, int &nb_good_candidates,
               void *data, int verbose_level),
         void *data,
         int verbose_level);
   void print();
   void early_test_func(
      long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   void unrank_point(int *v, long int rk);
   long int rank_point(int *v);
   void orbits_on_k_sets(
         poset_classification_control *Control,
         int k, long int *&orbit_reps,
         int &nb_orbits, int verbose_level);
   poset_classification *orbits_on_k_sets_compute(
         poset_classification_control *Control,
         int k, int verbose_level);
   void invoke_print_function(
         std::ostream &ost, int sz, long int *set);
};
```

```
// ##############################################################################
// upstep_work.cpp
// ##############################################################################


typedef struct coset_table_entry coset_table_entry;

//! auxiliary class to build a coset transversal for the automorphism group in th
e poset classification algorithm

struct coset_table_entry {
   int coset;
      // as in the loop in upstep_work::upstep_subspace_action
      // goes from 0 to degree - 1.

   int node; // = final_node as computed by recognize
   int ex; // = final_ex as computed by recognize
   int type; // = return value of recognize

   int nb_times_image_of_called;
   int nb_times_mult_called;
   int nb_times_invert_called;
   int nb_times_retrieve_called;
};


//! auxiliary class for the poset classification algorithm to deal with flag orbi
ts


class upstep_work {
public:
   poset_classification *gen;
   int size; // size = size of the object at prev
   int prev;
   int prev_ex;
   int cur;
   int nb_fusion_nodes;
   int nb_fuse_cur;
   int nb_ext_cur;
   int f_debug;
   int f_implicit_fusion;
   int f_indicate_not_canonicals;
   int mod_for_printing;
```

```
int pt;
int pt_orbit_len;

int *path; // [size + 1]
   // path[i] is the node that represents set[0,..,i-1]



poset_orbit_node *O_prev;
poset_orbit_node *O_cur;

data_structures_groups::group_container *G;
data_structures_groups::group_container *H;
ring_theory::longinteger_object go_G, go_H;

int coset;

int nb_cosets;
int nb_cosets_processed;
coset_table_entry *coset_table;

int *Elt1;
int *Elt2;
int *Elt3;


upstep_work();
~upstep_work();
void init(
      poset_classification *gen,
   int size,
   int prev,
   int prev_ex,
   int cur,
   int f_debug,
   int f_implicit_fusion,
   int f_indicate_not_canonicals,
   int verbose_level);
   // called from poset_classification::extend_node
void handle_extension(
      int &nb_fuse_cur, int &nb_ext_cur,
   int verbose_level);
   // called from poset_classification::extend_node
   // Calls handle_extension_fusion_type
   // or handle_extension_unprocessed_type
   //
```

```
   // Handles the extension ´cur_ex´ in node ´prev´.
   // We are extending a set of size ´size´
   // to a set of size ´size´ + 1.
   // Calls poset_orbit_node::init_extension_node for the
   // n e w node that is (possibly) created
void handle_extension_fusion_type(
      int verbose_level);
   // called from upstep_work::handle_extension
   // Handles the extension ´cur_ex´ in node ´prev´.
void handle_extension_unprocessed_type(int verbose_level);
   // called from upstep_work::handle_extension
   // calls init_extension_node
int init_extension_node(int verbose_level);
   // Called from upstep_work::handle_extension_unprocessed_type
   // Calls upstep_subspace_action or upstep_for_sets,
   // depending on the type of action
   // then changes the type of the extension to
   // EXTENSION_TYPE_EXTENSION

   // Establishes a n e w node at depth ´size´
   // (i.e., a set of size ´size´) as an extension
   // of a previous node (prev) at depth size - 1
   // with respect to a given point (pt).
   // This function is to be called for the next
   // free poset_orbit_node node which will
   // become the descendant of the previous node (prev).
   // the extension node corresponds to the point pt.
   // returns FALSE if the set is not canonical
   // (provided f_indicate_not_canonicals is TRUE)
int upstep_for_sets(int verbose_level);
   // This routine is called from upstep_work::init_extension_node
   // It is testing a set of size ´size´.
   // The newly added point is in gen->S[size - 1]
   // returns FALSE if the set is not canonical
   // (provided f_indicate_not_canonicals is TRUE)
//void print_level_extension_info_original_size();
void print_level_extension_info();
void print_level_extension_coset_info();

// upstep_work_subspace_action.cpp:
int upstep_subspace_action(int verbose_level);
   // This routine is called from upstep_work::init_extension_node
   // It computes coset_table.
   // It is testing a set of size ´size´.
   // The newly added point is in gen->S[size - 1]
   // The extension is initiated from node ´prev´
   // and from extension ´prev_ex´
```

```
      // The node ´prev´ is at depth ´size´ - 1
      // returns FALSE if the set is not canonical
      // (provided f_indicate_not_canonicals is TRUE)


   // upstep_work_trace.cpp:

   trace_result recognize(
      int &final_node, int &final_ex, int f_tolerant,
      int verbose_level);
   trace_result recognize_recursion(
      int lvl, int current_node, int &final_node, int &final_ex,
      int f_tolerant,
      int verbose_level);
   trace_result handle_last_level(
      int lvl, int current_node, int current_extension, int pt0,
      int &final_node, int &final_ex,
      int verbose_level);
   trace_result start_over(
      int lvl, int current_node,
      int &final_node, int &final_ex,
      int f_tolerant, int verbose_level);
};


}}}


#endif /* ORBITER_SRC_LIB_CLASSIFICATION_POSET_CLASSIFICATION_POSET_CLASSIFICATIO
N_H_ */
```

6.6 Set Stabilizer

```
/*
 * set_stabilizer.h
 *
 *  Created on: Jun 27, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_GROUP_ACTIONS_SET_STABILIZER_SET_STABILIZER_H_
#define SRC_LIB_GROUP_ACTIONS_SET_STABILIZER_SET_STABILIZER_H_

namespace orbiter {
namespace layer4_classification {
namespace set_stabilizer {



// ##############################################################################
// compute_stabilizer.cpp
// ##############################################################################

//! to compute the set-stabilizer

class compute_stabilizer {

public:


   substructure_stats_and_selection *SubSt;

   actions::action *A_on_the_set;
      // only used to print the induced action on the set
      // of the set stabilizer

   groups::sims *Stab; // the stabilizer of the original set


   ring_theory::longinteger_object stab_order, new_stab_order;
   int nb_times_orbit_count_does_not_match_up;
   int backtrack_nodes_first_time;
   int backtrack_nodes_total_in_loop;

   stabilizer_orbits_and_types *Stab_orbits;
```

```
actions::action *A_induced;
   // the action on Stab_orbits->interesting_points[]

ring_theory::longinteger_object induced_go, K_go;

int *transporter_witness;
int *transporter1;
int *transporter2;
int *T1, *T1v;
int *T2;

groups::sims *Kernel_original;
groups::sims *K; // kernel for building up Stab



groups::sims *Aut;
groups::sims *Aut_original;
ring_theory::longinteger_object ago;
ring_theory::longinteger_object ago1;
ring_theory::longinteger_object target_go;


//union_find_on_k_subsets *U;


long int *Canonical_form_input;
   // [nb_interesting_subsets_reduced * reduced_set_size]
long int *Canonical_forms;
   // [nb_interesting_subsets_reduced * reduced_set_size]
int *Canonical_form_transporter;
   // [nb_interesting_subsets_reduced * A_induced->elt_size_in_int]

int nb_interesting_subsets_rr;
long int *interesting_subsets_rr;


compute_stabilizer();
~compute_stabilizer();

void init(
      substructure_stats_and_selection *SubSt,
      long int *canonical_pts,
      int verbose_level);
```

```
   void compute_automorphism_group(int verbose_level);
   void compute_automorphism_group_handle_case(
         int cnt2, int verbose_level);
   void setup_stabilizer(groups::sims *Stab0, int verbose_level);
   void restricted_action_on_interesting_points(int verbose_level);
   void compute_canonical_form(int verbose_level);
   void compute_canonical_form_handle_case(
         int cnt, int verbose_level);
   void compute_canonical_set(
         long int *set_in, long int *set_out, int sz,
         int *transporter, int verbose_level);
   void compute_canonical_set_and_group(
         long int *set_in, long int *set_out, int sz,
         int *transporter,
         groups::sims *&stab, int verbose_level);
   void update_stabilizer(int verbose_level);
   void add_automorphism(int verbose_level);
   void retrieve_automorphism(int verbose_level);
   void make_canonical_second_set(int verbose_level);
   void report(std::ostream &ost);
   void print_canonical_sets();

};





// ##############################################################################
// stabilizer_orbits_and_types.cpp
// ##############################################################################




//! orbits of the stabilizer of the substructure and orbit types



class stabilizer_orbits_and_types {

public:
   compute_stabilizer *CS;

   groups::strong_generators *selected_set_stab_gens;
   groups::sims *selected_set_stab;
```

```
int reduced_set_size; // = set_size - level




long int *reduced_set1; // [set_size]
long int *reduced_set2; // [set_size]
long int *reduced_set1_new_labels; // [set_size]
long int *reduced_set2_new_labels; // [set_size]
long int *canonical_set1; // [set_size]
long int *canonical_set2; // [set_size]

int *elt1, *Elt1, *Elt1_inv, *new_automorphism, *Elt4;
int *elt2, *Elt2;
int *transporter0; // = elt1 * elt2

ring_theory::longinteger_object go_G;

groups::schreier *Schreier;
int nb_orbits;
int *orbit_count1; // [nb_orbits]
int *orbit_count2; // [nb_orbits]


int minimal_orbit_pattern_idx;
int nb_interesting_subsets_reduced;
long int *interesting_subsets_reduced;

int *Orbit_patterns; // [nb_interesting_subsets * nb_orbits]


int *orbit_to_interesting_orbit; // [nb_orbits]

int nb_interesting_orbits;
int *interesting_orbits; // [nb_interesting_orbits]

int nb_interesting_points; // sum of orbit length of interesting orbits
long int *interesting_points; // [nb_interesting_points]
// Note: Interesting points are sorted within each orbit.
// Otherwise, it would not be possible to compute a canonical form.

int *interesting_orbit_first; // [nb_interesting_orbits]
int *interesting_orbit_len; // [nb_interesting_orbits]

int local_idx1, local_idx2;
```

```
   stabilizer_orbits_and_types();
   ~stabilizer_orbits_and_types();
   void init(
         compute_stabilizer *CS,
         int verbose_level);
   void compute_stabilizer_orbits_and_find_minimal_pattern(
         int verbose_level);
   // uses selected_set_stab_gens to compute orbits on points in action A2
   void save_interesting_subsets_reduced(
         int stage, int verbose_level);
   void find_orbit_pattern(
         int cnt, int *transp, int verbose_level);
   // computes transporter to transp
   void find_interesting_orbits(int verbose_level);
   void compute_local_labels(
         long int *set_in, long int *set_out, int sz,
         int verbose_level);
   void map_subset_and_compute_local_labels(
         int cnt, int verbose_level);
   void map_reduced_set_and_do_orbit_counting(
         int cnt,
         long int subset_idx, int *transporter, int verbose_level);
      // computes orbit_count1[]
   int check_orbit_count();
   void print_orbit_count(int f_both);
   void print_minimal_orbit_pattern();

};


// ##############################################################################
// substructure_classifier.cpp
// ##############################################################################



//! classification of substructures




class substructure_classifier {
public:
```

```
   std::string fname_base_out;
   int substructure_size;

   poset_classification::poset_classification *PC;
   poset_classification::poset_classification_control *Control;
   actions::action *A;
   actions::action *A2;
   poset_classification::poset_with_group_action *Poset;
   int nb_orbits;


   substructure_classifier();
   ~substructure_classifier();
   void classify_substructures(
         std::string &fname_base_out,
         actions::action *A,
         actions::action *A2,
         groups::strong_generators *gens,
         int substructure_size,
         int verbose_level);
   void set_stabilizer_in_any_space(
         actions::action *A, actions::action *A2,
         groups::strong_generators *Strong_gens,
         int intermediate_subset_size,
         std::string &fname_mask, int nb, std::string &column_label,
         std::string &fname_out,
         int verbose_level);
   void set_stabilizer_of_set(
         std::string &fname_out,
         int cnt, int nb, int row,
         long int *pts,
         int nb_pts,
         long int *canonical_pts,
         int verbose_level);
   void handle_orbit(
         substructure_stats_and_selection *SubSt,
         long int *canonical_pts,
         int *transporter_to_canonical_form,
         groups::strong_generators *&Gens_stabilizer_original_set,
         int verbose_level);


};
```

```
// ##############################################################################
// substructure_stats_and_selection.cpp
// ##############################################################################



//! analyzing the substructures of a given set




class substructure_stats_and_selection {
public:

   std::string fname_case_out;

   substructure_classifier *SubC;

   long int *Pts;
   int nb_pts;


   // computed by SubC->PC->trace_all_k_subsets_and_compute_frequencies:
   int nCk;
   int *isotype; // [nCk]
   int *orbit_frequencies; // [nb_orbits]
   int nb_orbits;
   data_structures::tally *T;
   // for orbit_frequencies[]


   data_structures::set_of_sets *SoS;
   int *types;
   int nb_types;
   int selected_type;
   int selected_orbit;
   int selected_frequency;

   long int *interesting_subsets;
      // [selected_frequency]
   int nb_interesting_subsets;
      // interesting_subsets are the lvl-subsets of the given set
      // which are of the chosen type.
      // There is nb_interesting_subsets of them.
```

```
   groups::strong_generators *gens;
      // generators for the selected canonical subset
   //int *transporter_to_canonical_form;
   //strong_generators *Gens_stabilizer_original_set;


   substructure_stats_and_selection();
   ~substructure_stats_and_selection();
   void init(
         std::string &fname_case_out,
         substructure_classifier *SubC,
         long int *Pts,
         int nb_pts,
         int verbose_level);

};




}}}




#endif /* SRC_LIB_GROUP_ACTIONS_SET_STABILIZER_SET_STABILIZER_H_ */
```

6.7 Solvers

```
// solver.h
//
// Anton Betten
//
// moved here from top_level.h: July 28, 2018
// top_level started:  September 23 2010
// based on global.h, which was taken from reader.h: 3/22/09



#ifndef ORBITER_SRC_LIB_TOP_LEVEL_SOLVER_SOLVER_H_
#define ORBITER_SRC_LIB_TOP_LEVEL_SOLVER_SOLVER_H_


namespace orbiter {
namespace layer4_classification {
namespace solvers_package {


// ##############################################################################
// exact_cover.cpp
// ##############################################################################


//! exact cover problems arising with the lifting of combinatorial objects


class exact_cover {
public:

   std::string input_prefix;
   std::string output_prefix;
   std::string solution_prefix;
   std::string base_fname;

   std::string fname_solutions;
   std::string fname_statistics;

   void *user_data;

   actions::action *A_base;
   actions::action *A_on_blocks;


   void (*prepare_function_new)(
         exact_cover *E, int starter_case,
```

```
   long int *candidates, int nb_candidates,
   groups::strong_generators *Strong_gens,
   solvers::diophant *&Dio, long int *&col_label,
   int &f_ruled_out,
   int verbose_level);



void (*early_test_func)(
      long int *S, int len,
   long int *candidates, int nb_candidates,
   long int *good_candidates, int &nb_good_candidates,
   void *data, int verbose_level);
void *early_test_func_data;

int f_has_solution_test_func;
int (*solution_test_func)(
      exact_cover *EC, long int *S, int len,
   void *data, int verbose_level);
void *solution_test_func_data;

int f_has_late_cleanup_function;
void (*late_cleanup_function)(
      exact_cover *E,
   int starter_case, int verbose_level);


int target_size;

int f_lex;

int f_split;
int split_r;
int split_m;

int f_single_case;
int single_case;

int starter_size;
int starter_nb_cases;
long int *starter; // [starter_size]

int f_randomized;
std::string random_permutation_fname;
int *random_permutation;

exact_cover();
```

```
~exact_cover();
void null();
void freeself();
void init_basic(
      void *user_data,
      actions::action *A_base,
      actions::action *A_on_blocks,
   int target_size, int starter_size,
   std::string &input_prefix,
   std::string &output_prefix,
   std::string &solution_prefix,
   std::string &base_fname,
   int f_lex,
   int verbose_level);

void init_early_test_func(
   void (*early_test_func)(long int *S, int len,
         long int *candidates, int nb_candidates,
         long int *good_candidates, int &nb_good_candidates,
      void *data, int verbose_level),
   void *early_test_func_data,
   int verbose_level);
void init_prepare_function_new(
   void (*prepare_function_new)(
         solvers_package::exact_cover *E, int starter_case,
         long int *candidates, int nb_candidates,
         groups::strong_generators *Strong_gens,
      solvers::diophant *&Dio, long int *&col_label,
      int &f_ruled_out,
      int verbose_level),
   int verbose_level);
void set_split(int split_r, int split_m, int verbose_level);
void set_single_case(int single_case, int verbose_level);
void randomize(
      std::string &random_permutation_fname,
      int verbose_level);
void add_solution_test_function(
   int (*solution_test_func)(
         exact_cover *EC, long int *S, int len,
   void *data, int verbose_level),
   void *solution_test_func_data,
   int verbose_level);
void add_late_cleanup_function(
   void (*late_cleanup_function)(
         exact_cover *E,
   int starter_case, int verbose_level)
   );
```

```
   void compute_liftings_new(
         int f_solve, int f_save, int f_read_instead,
      int f_draw_system,
      std::string &fname_system,
      int f_write_tree,
      std::string &fname_tree, int verbose_level);
   void compute_liftings_single_case_new(
         int starter_case,
      int f_solve, int f_save, int f_read_instead,
      int &nb_col,
      long int *&Solutions, int &sol_length, int &nb_sol,
      int &nb_backtrack, int &dt,
      int f_draw_system, std::string &fname_system,
      int f_write_tree, std::string &fname_tree,
      int verbose_level);
   void lexorder_test(
         long int *live_blocks2, int &nb_live_blocks2,
         data_structures_groups::vector_ge *stab_gens,
      int verbose_level);
};




// ##############################################################################
// exact_cover_arguments.cpp
// ##############################################################################


//! command line arguments to control the lifting via exact cover



class exact_cover_arguments {
public:
   actions::action *A;
   actions::action *A2;
   void *user_data;
   int f_has_base_fname;
   std::string base_fname;
   int f_has_input_prefix;
   std::string input_prefix;
   int f_has_output_prefix;
   std::string output_prefix;
   int f_has_solution_prefix;
   std::string solution_prefix;
   int f_lift;
```

```
   int f_starter_size;
   int starter_size;
   int target_size;
   int f_lex;
   int f_split;
   int split_r;
   int split_m;
   int f_solve;
   int f_save;
   int f_read;
   int f_draw_system;
   std::string fname_system;
   int f_write_tree;
   std::string fname_tree;
   void (*prepare_function_new)(
         exact_cover *E, int starter_case,
      long int *candidates, int nb_candidates,
      groups::strong_generators *Strong_gens,
      solvers::diophant *&Dio, long int *&col_label,
      int &f_ruled_out,
      int verbose_level);
   void (*early_test_function)(
         long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      void *data, int verbose_level);
   void *early_test_function_data;
   int f_has_solution_test_function;
   int (*solution_test_func)(
         exact_cover *EC, long int *S, int len,
      void *data, int verbose_level);
   void *solution_test_func_data;
   int f_has_late_cleanup_function;
   void (*late_cleanup_function)(exact_cover *EC,
      int starter_case, int verbose_level);

   int f_randomized;
   std::string random_permutation_fname;

   exact_cover_arguments();
   ~exact_cover_arguments();
   void null();
   void freeself();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void compute_lifts(int verbose_level);
};
```

```
}}}


#endif /* ORBITER_SRC_LIB_TOP_LEVEL_SOLVER_SOLVER_H_ */
```

Chapter 7

Layer 5 – Applications

7.1 Main Header File

```
// top_level.h
//
// Anton Betten
//
// started:  September 23 2010
//
// based on global.h, which was taken from reader.h: 3/22/09



#ifndef ORBITER_SRC_LIB_TOP_LEVEL_TOP_LEVEL_H_
#define ORBITER_SRC_LIB_TOP_LEVEL_TOP_LEVEL_H_



using namespace orbiter::layer1_foundations;
using namespace orbiter::layer2_discreta;
using namespace orbiter::layer3_group_actions;
using namespace orbiter::layer4_classification;


namespace orbiter {

//! classes for combinatorial objects and their classification

namespace layer5_applications {



//! Applications in algebra and number theory
```

```
namespace apps_algebra {

   // algebra_and_number_theory
   class action_on_forms_activity_description;
   class action_on_forms_activity;
   class action_on_forms_description;
   class action_on_forms;
   class algebra_global_with_action;
   class any_group;
   class character_table_burnside;
   class element_processing_description;
   class group_modification_description;
   class group_theoretic_activity_description;
   class group_theoretic_activity;
   class modified_group_create;
   class orbit_cascade;
   class orbits_activity_description;
   class orbits_activity;
   class orbits_create_description;
   class orbits_create;
   class orbits_on_polynomials;
   class orbits_on_subspaces;
   class polynomial_ring_activity;
   class vector_ge_builder;
   class young;

}

//! Applications in coding theory



namespace apps_coding_theory {

   class code_modification_description;
   class coding_theoretic_activity_description;
   class coding_theoretic_activity;
   class crc_process_description;
   class crc_process;
   class create_code_description;
   class create_code;
}


//! Classification problems in combinatorics, including design theory
```

```
namespace apps_combinatorics {

   // combinatorics
   class boolean_function_classify;
   class combinatorial_object_activity_description;
   class combinatorial_object_activity;
   class combinatorics_global;
   class delandtsheer_doyen_description;
   class delandtsheer_doyen;
   class design_activity_description;
   class design_activity;
   class design_create_description;
   class design_create;
   class design_tables;
   class difference_set_in_heisenberg_group;
   class flag_orbits_incidence_structure;
   class hadamard_classify;
   class hall_system_classify;
   class large_set_activity_description;
   class large_set_activity;
   class large_set_classify;
   class large_set_was_activity_description;
   class large_set_was_activity;
   class large_set_was_description;
   class large_set_was;
   class object_with_properties;
   class regular_linear_space_description;
   class regular_ls_classify;
   class tactical_decomposition;

}

//! Geometry

namespace apps_geometry {

   // geometry
   class arc_generator_description;
   class arc_generator;
   class arc_lifting_simeon;
   class choose_points_or_lines;
   class classify_cubic_curves;
   class cubic_curve_with_action;
   class hermitian_spread_classify;
   class linear_set_classify;
   class ovoid_classify_description;
   class ovoid_classify;
```

```
   class polar;
   class search_blocking_set;
   class singer_cycle;
   class tensor_classify;
   class top_level_geometry_global;

}

//! Graph theory

namespace apps_graph_theory {

   // graph_theory.h:
   class cayley_graph_search;
   class create_graph_description;
   class create_graph;
   class graph_classification_activity_description;
   class graph_classification_activity;
   class graph_classify_description;
   class graph_classify;
   class graph_modification_description;
   class graph_theoretic_activity_description;
   class graph_theoretic_activity;
   class graph_theory_apps;

}

//! Orbiter command line, orbiter dash code, symbol definitions and activities

namespace user_interface {

   // interfaces
   class activity_description;
   class interface_algebra;
   class interface_coding_theory;
   class interface_combinatorics;
   class interface_cryptography;
   class interface_povray;
   class interface_projective;
   class interface_symbol_table;
   class interface_toolkit;
   class orbiter_command;
   class orbiter_top_level_session;
   class symbol_definition;

}
```

```
//! Applications in orthogonal geometry

namespace orthogonal_geometry_applications {


   // orthogonal
   class blt_set_activity_description;
   class blt_set_activity;
   class blt_set_classify_activity_description;
   class blt_set_classify_description;
   class blt_set_classify;
   class BLT_set_create_description;
   class BLT_set_create;
   class blt_set_with_action;
   class flock;
   class orthogonal_space_activity_description;
   class orthogonal_space_activity;
   class orthogonal_space_with_action_description;
   class orthogonal_space_with_action;

}

//! packings in projective space

namespace packings {

   // packings:
   class invariants_packing;
   class packing_classify;
   class packing_invariants;
   class packing_long_orbits_description;
   class packing_long_orbits;
   class packing_was_activity_description;
   class packing_was_activity;
   class packing_was_description;
   class packing_was_fixpoints_activity_description;
   class packing_was_fixpoints_activity;
   class packing_was_fixpoints;
   class packing_was;
   class packings_global;
   class regular_packing;

}


//! Applications in projective space
```

```
namespace projective_geometry {

   // projective_space.h:
   class canonical_form_classifier_description;
   class canonical_form_classifier;
   class canonical_form_nauty;
   class canonical_form_of_variety;
   class canonical_form_substructure;
   class object_in_projective_space_with_action;
   class projective_space_activity_description;
   class projective_space_activity;
   class projective_space_globals;
   class projective_space_with_action_description;
   class projective_space_with_action;
   class quartic_curve_object;

}

//! Classification of finite semifields


namespace semifields {


   // semifield
   class semifield_classify_description;
   class semifield_classify_with_substructure;
   class semifield_classify;
   class semifield_downstep_node;
   class semifield_flag_orbit_node;
   class semifield_level_two;
   class semifield_lifting;
   class semifield_substructure;
   class semifield_trace;
   class trace_record;

}


//! Spreads in projective space

namespace spreads {

   // spreads:
   class recoordinatize;
   class spread_activity_description;
```

```
   class spread_activity;
   class spread_classify_activity_description;
   class spread_classify_activity;
   class spread_classify_description;
   class spread_classify;
   class spread_create_description;
   class spread_create;
   class spread_lifting;
   class spread_table_activity_description;
   class spread_table_activity;
   class spread_table_with_selection;
   class translation_plane_activity_description;
   class translation_plane_activity;

}


//! cubic surfaces, quartic curves, etc.

namespace applications_in_algebraic_geometry {


   //! classes related to plane quartic curves with 28 bitangents

   namespace quartic_curves {

      // surfaces/quartic curves
      class quartic_curve_activity_description;
      class quartic_curve_activity;
      class quartic_curve_create_description;
      class quartic_curve_create;
      class quartic_curve_domain_with_action;
      class quartic_curve_from_surface;
      class quartic_curve_object_with_action;

   }

   //! cubic surfaces and related six-arcs and trihedral pairs

   namespace cubic_surfaces_and_arcs {

      // surfaces/surfaces_and_arcs:
      class arc_lifting;
      class arc_orbits_on_pairs;
      class arc_partition;
      class classify_trihedral_pairs;
      class six_arcs_not_on_a_conic;
```

```
      class surface_classify_using_arc;
      class surface_create_by_arc_lifting;
      class surfaces_arc_lifting_definition_node;
      class surfaces_arc_lifting_trace;
      class surfaces_arc_lifting_upstep;
      class surfaces_arc_lifting;
      class trihedral_pair_with_action;
      // pointer types:
      typedef class surfaces_arc_lifting_trace psurfaces_arc_lifting_trace;

   }

   //! cubic surfaces and related Schlaefli double-sixes

   namespace cubic_surfaces_and_double_sixes {

      // surfaces/surfaces_and_double_sixes:
      class classification_of_cubic_surfaces_with_double_sixes_activity_description
;
      class classification_of_cubic_surfaces_with_double_sixes_activity;
      class classify_double_sixes;
      class classify_five_plus_one;
      class surface_classify_wedge;

   }

   //! cubic surfaces in general

   namespace cubic_surfaces_in_general {

      // surfaces/surfaces_general:
      class cubic_surface_activity_description;
      class cubic_surface_activity;
      class surface_clebsch_map;
      class surface_create_description;
      class surface_create;
      class surface_domain_high_level;
      class surface_object_with_action;
      class surface_study;
      class surface_with_action;

   }

}

#define Get_object_of_type_any_group(label) user_interface::The_Orbiter_top_level
_session->get_object_of_type_any_group(label)
```

```
#define Get_vector_or_set(label, set, sz) user_interface::The_Orbiter_top_level_s
ession->get_vector_or_set(label, set, sz, 0)


//#define Get_object_of_type_finite_field(label) user_interface::The_Orbiter_top_
level_session->get_object_of_type_finite_field(label)
// use Get_finite_field() instead

#define Get_object_of_type_any_group(label) user_interface::The_Orbiter_top_level
_session->get_object_of_type_any_group(label)
#define Get_object_of_type_spread(label) user_interface::The_Orbiter_top_level_se
ssion->get_object_of_type_spread(label)
//#define Get_object_of_type_ring(label) user_interface::The_Orbiter_top_level_se
ssion->get_object_of_type_ring(label)
#define Get_object_of_type_poset_classification_control(label) user_interface::Th
e_Orbiter_top_level_session->get_object_of_type_poset_classification_control(labe
l)
#define Get_object_of_type_vector_ge(label) user_interface::The_Orbiter_top_level
_session->get_object_of_type_vector_ge(label)
#define Get_object_of_projective_space(label) user_interface::The_Orbiter_top_lev
el_session->get_object_of_type_projective_space(label)
#define Get_object_of_cubic_surface(label) user_interface::The_Orbiter_top_level_
session->get_object_of_type_cubic_surface(label)
#define Get_object_of_type_code(label) user_interface::The_Orbiter_top_level_sess
ion->get_object_of_type_code(label)
#define Get_orthogonal_space(label) user_interface::The_Orbiter_top_level_session
->get_orthogonal_space(label)



}}


#include "./algebra_and_number_theory/tl_algebra_and_number_theory.h"
#include "./coding_theory_apps/coding_theory_apps.h"
#include "./combinatorics/tl_combinatorics.h"
#include "./geometry/tl_geometry.h"
#include "./graph_theory/tl_graph_theory.h"
#include "./interfaces/interfaces.h"
#include "./orthogonal/tl_orthogonal.h"
#include "./projective_space/projective_space.h"
#include "./semifields/semifields.h"
#include "./spreads/spreads.h"
#include "./packings/packings.h"
#include "./surfaces/quartic_curves/quartic_curves.h"
#include "./surfaces/surfaces_and_arcs/surfaces_and_arcs.h"
#include "./surfaces/surfaces_and_double_sixes/surfaces_and_double_sixes.h"
```

```
#include "./surfaces/surfaces_general/surfaces_general.h"


#endif /* ORBITER_SRC_LIB_TOP_LEVEL_TOP_LEVEL_H_ */
```

7.2 Algebra and Number Theory

```
// tl_algebra_and_number_theory.h
//
// Anton Betten
//
// moved here from top_level.h: July 28, 2018
// top_level started:  September 23 2010
// based on global.h, which was taken from reader.h: 3/22/09


#ifndef ORBITER_SRC_LIB_TOP_LEVEL_ALGEBRA_AND_NUMBER_THEORY_TL_ALGEBRA_AND_NUMBER
_THEORY_H_
#define ORBITER_SRC_LIB_TOP_LEVEL_ALGEBRA_AND_NUMBER_THEORY_TL_ALGEBRA_AND_NUMBER
_THEORY_H_


namespace orbiter {
namespace layer5_applications {
namespace apps_algebra {


// ##############################################################################
// action_on_forms_activity_description.cpp
// ##############################################################################


//! description of an action of forms


class action_on_forms_activity_description {

public:

   int f_algebraic_normal_form;
   std::string algebraic_normal_form_input;

   int f_orbits_on_functions;
   std::string orbits_on_functions_input;

   int f_associated_set_in_plane;
   std::string associated_set_in_plane_input;

   int f_differential_uniformity;
   std::string differential_uniformity_input;


   action_on_forms_activity_description();
```

```
   ~action_on_forms_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// action_on_forms_activity.cpp
// ##############################################################################


//! perform an activity associated with an action on forms

class action_on_forms_activity {
public:
   action_on_forms_activity_description *Descr;

   apps_algebra::action_on_forms *AF;



   action_on_forms_activity();
   ~action_on_forms_activity();
   void init(
         action_on_forms_activity_description *Descr,
         apps_algebra::action_on_forms *AF,
         int verbose_level);
   void perform_activity(int verbose_level);
   void do_algebraic_normal_form(int verbose_level);
   void do_orbits_on_functions(int verbose_level);
   void do_associated_set_in_plane(int verbose_level);
   void do_differential_uniformity(int verbose_level);

};



// ##############################################################################
// action_on_forms_description.cpp
// ##############################################################################


//! description of an action of forms
```

```
class action_on_forms_description {

public:

   int f_space;
   std::string space_label;

   int f_degree;
   int degree;


   action_on_forms_description();
   ~action_on_forms_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};




// ##############################################################################
// action_on_forms.cpp
// ##############################################################################


//! an action of forms


class action_on_forms {

public:
   action_on_forms_description *Descr;

   std::string prefix;
   std::string label_txt;
   std::string label_tex;

   int q;
   field_theory::finite_field *F;

   int f_semilinear;
```

```
   projective_geometry::projective_space_with_action *PA;

   combinatorics::polynomial_function_domain *PF;

   actions::action *A_on_poly;

   int f_has_group;
   groups::strong_generators *Sg;




   action_on_forms();
   ~action_on_forms();
   void create_action_on_forms(
         action_on_forms_description *Descr,
         int verbose_level);
   void orbits_on_functions(
         int *The_functions, int nb_functions, int len,
         int verbose_level);
   void orbits_on_equations(
         int *The_equations, int nb_equations, int len,
         groups::schreier *&Orb,
         actions::action *&A_on_equations,
         int verbose_level);
   void associated_set_in_plane(
         int *func, int len,
         long int *&Rk, int verbose_level);
   void differential_uniformity(
         int *func, int len, int verbose_level);

};


// ##############################################################################
// algebra_global_with_action.cpp
// ##############################################################################

//! group theoretic functions which require an action


class algebra_global_with_action {
public:
   void orbits_under_conjugation(
         long int *the_set, int set_size, groups::sims *S,
         groups::strong_generators *SG,
         data_structures_groups::vector_ge *Transporter,
```

```
      int verbose_level);
void create_subgroups(
      groups::strong_generators *SG,
      long int *the_set, int set_size, groups::sims *S,
      actions::action *A_conj,
      groups::schreier *Classes,
      data_structures_groups::vector_ge *Transporter,
      int verbose_level);
void compute_orbit_of_set(
      long int *the_set, int set_size,
      actions::action *A1, actions::action *A2,
      data_structures_groups::vector_ge *gens,
      std::string &label_set,
      std::string &label_group,
      long int *&Table,
      int &orbit_length,
      int verbose_level);
void conjugacy_classes_based_on_normal_forms(
      actions::action *A,
      groups::sims *override_Sims,
      std::string &label,
      std::string &label_tex,
      int verbose_level);
void classes_GL(
      field_theory::finite_field *F,
      int d, int f_no_eigenvalue_one, int verbose_level);
void do_normal_form(int q, int d,
      int f_no_eigenvalue_one, int *data, int data_sz,
      int verbose_level);
void do_identify_one(int q, int d,
      int f_no_eigenvalue_one, int elt_idx,
      int verbose_level);
void do_identify_all(int q, int d,
      int f_no_eigenvalue_one, int verbose_level);
void do_random(
      int q, int d,
      int f_no_eigenvalue_one, int verbose_level);
void group_table(
      int q, int d, int f_poly, std::string &poly,
      int f_no_eigenvalue_one, int verbose_level);
void centralizer_brute_force(int q, int d,
      int elt_idx, int verbose_level);
void centralizer(int q, int d,
      int elt_idx, int verbose_level);
// creates a finite_field, and two actions
// using init_projective_group and init_general_linear_group
void centralizer(int q, int d, int verbose_level);
```

```
void compute_regular_representation(
      actions::action *A, groups::sims *S,
      data_structures_groups::vector_ge *SG,
      int *&perm, int verbose_level);
void presentation(actions::action *A,
      groups::sims *S, int goi,
      data_structures_groups::vector_ge *gens,
   int *primes, int verbose_level);

void do_eigenstuff(
      field_theory::finite_field *F,
      int size, int *Data, int verbose_level);
void A5_in_PSL_(int q, int verbose_level);
void A5_in_PSL_2_q(int q,
      layer2_discreta::typed_objects::discreta_matrix & A,
      layer2_discreta::typed_objects::discreta_matrix & B,
      layer2_discreta::typed_objects::domain *dom_GFq, int verbose_level);
void A5_in_PSL_2_q_easy(int q,
      layer2_discreta::typed_objects::discreta_matrix & A,
      layer2_discreta::typed_objects::discreta_matrix & B,
      layer2_discreta::typed_objects::domain *dom_GFq,
      int verbose_level);
void A5_in_PSL_2_q_hard(int q,
      layer2_discreta::typed_objects::discreta_matrix & A,
      layer2_discreta::typed_objects::discreta_matrix & B,
      layer2_discreta::typed_objects::domain *dom_GFq,
      int verbose_level);
int proj_order(layer2_discreta::typed_objects::discreta_matrix &A);
void trace(layer2_discreta::typed_objects::discreta_matrix &A,
      layer2_discreta::typed_objects::discreta_base &tr);
void elementwise_power_int(
      layer2_discreta::typed_objects::discreta_matrix &A, int k);
int is_in_center(
      layer2_discreta::typed_objects::discreta_matrix &B);
void matrix_convert_to_numerical(
      layer2_discreta::typed_objects::discreta_matrix &A, int *AA, int q);


void young_symmetrizer(
      int n, int verbose_level);
void young_symmetrizer_sym_4(int verbose_level);
void report_tactical_decomposition_by_automorphism_group(
      std::ostream &ost,
      geometry::projective_space *P,
      actions::action *A_on_points, actions::action *A_on_lines,
      groups::strong_generators *gens, int size_limit_for_printing,
      int verbose_level);
```

```
   void linear_codes_with_bounded_minimum_distance(
         poset_classification::poset_classification_control *Control,
         groups::linear_group *LG,
         int d, int target_depth, int verbose_level);
   void permutation_representation_of_element(
         actions::action *A,
         std::string &element_description,
         int verbose_level);
   void relative_order_vector_of_cosets(
         actions::action *A, groups::strong_generators *SG,
         data_structures_groups::vector_ge *cosets,
         int *&relative_order_table, int verbose_level);
   void representation_on_polynomials(
         groups::linear_group *LG,
         int degree_of_poly,
         int verbose_level);

   void do_eigenstuff_with_coefficients(
         field_theory::finite_field *F,
         int n, std::string &coeffs_text, int verbose_level);
   void do_eigenstuff_from_file(
         field_theory::finite_field *F,
         int n, std::string &fname, int verbose_level);

   void orbits_on_points(
         actions::action *A2,
         groups::strong_generators *Strong_gens,
         int f_load_save,
         std::string &prefix,
         groups::orbits_on_something *&Orb,
         int verbose_level);
   void find_singer_cycle(any_group *Any_group,
         actions::action *A1, actions::action *A2,
         int verbose_level);
   void search_element_of_order(any_group *Any_group,
         actions::action *A1, actions::action *A2,
         int order, int verbose_level);
   void find_standard_generators(any_group *Any_group,
         actions::action *A1, actions::action *A2,
         int order_a, int order_b, int order_ab, int verbose_level);

};



// ##############################################################################
// any_group.cpp
```

```
// ##############################################################################

//! a wrapper for linear_group and permutation_group_create

class any_group {

public:

   int f_linear_group;
   groups::linear_group *LG;

   int f_permutation_group;
   groups::permutation_group_create *PGC;

   int f_modified_group;
   modified_group_create *MGC;

   actions::action *A_base;
   actions::action *A;

   std::string label;
   std::string label_tex;

   groups::strong_generators *Subgroup_gens;
   groups::sims *Subgroup_sims;

   any_group();
   ~any_group();
   void init_linear_group(
         groups::linear_group *LG, int verbose_level);
   void init_permutation_group(
         groups::permutation_group_create *PGC,
         int verbose_level);
   void init_modified_group(
         modified_group_create *MGC, int verbose_level);
   void create_latex_report(
         graphics::layered_graph_draw_options *O,
         int f_sylow, int f_group_table, int f_classes,
         int verbose_level);
   void export_group_table(int verbose_level);
   void do_export_orbiter(
         actions::action *A2, int verbose_level);
   void do_export_gap(int verbose_level);
   void do_export_magma(int verbose_level);
   void do_canonical_image_GAP(
         std::string &input_set, int verbose_level);
   void create_group_table(
```

```
      int *&Table, long int &n, int verbose_level);
void normalizer(int verbose_level);
void centralizer(
      std::string &element_label,
      std::string &element_description_text,
      int verbose_level);
void permutation_representation_of_element(
      std::string &element_description_text,
      int verbose_level);
void normalizer_of_cyclic_subgroup(
      std::string &element_label,
      std::string &element_description_text,
      int verbose_level);
void do_find_subgroups(
      int order_of_subgroup,
      int verbose_level);
void print_elements(int verbose_level);
void print_elements_tex(int f_with_permutation,
      int f_override_action, actions::action *A_special,
      int verbose_level);
void order_of_products_of_elements_by_rank(
      std::string &Elements_text,
      int verbose_level);
void save_elements_csv(
      std::string &fname, int verbose_level);
void export_inversion_graphs(
      std::string &fname, int verbose_level);
void multiply_elements_csv(
      std::string &fname1,
      std::string &fname2,
      std::string &fname3,
      int f_column_major_ordering,
      int verbose_level);
void apply_elements_to_set_csv(
      std::string &fname1,
      std::string &fname2,
      std::string &set_text,
      int verbose_level);
void random_element(
      std::string &elt_label, int verbose_level);
void element_rank(
      std::string &elt_data, int verbose_level);
void element_unrank(
      std::string &rank_string, int verbose_level);
void conjugacy_class_of(
      std::string &label, std::string &rank_string,
      int verbose_level);
```

```
void do_reverse_isomorphism_exterior_square(int verbose_level);

void orbits_on_set_system_from_file(
      std::string &fname_csv,
      int number_of_columns, int first_column,
      int verbose_level);
void orbits_on_set_from_file(
      std::string &fname_csv, int verbose_level);
void orbit_of(int point_idx, int verbose_level);
void orbits_on_points(
      groups::orbits_on_something *&Orb, int verbose_level);

void create_latex_report_for_permutation_group(
      graphics::layered_graph_draw_options *O,
      int verbose_level);
void create_latex_report_for_modified_group(
      graphics::layered_graph_draw_options *O,
      int verbose_level);
groups::strong_generators *get_strong_generators();
int is_subgroup_of(
      any_group *AG_secondary, int verbose_level);
void set_of_coset_representatives(
      any_group *AG_secondary,
      data_structures_groups::vector_ge *&coset_reps,
      int verbose_level);
void report_coset_reps(
      data_structures_groups::vector_ge *coset_reps,
      int verbose_level);
void print_given_elements_tex(
      std::string &label_of_elements,
      int *element_data, int nb_elements,
      int f_with_permutation,
      int f_with_fix_structure,
      int verbose_level);
void process_given_elements(
      std::string &label_of_elements,
      int *element_data, int nb_elements,
      int verbose_level);
void apply_isomorphism_wedge_product_4to6(
      std::string &label_of_elements,
      int *element_data, int nb_elements,
      int verbose_level);
void order_of_products_of_pairs(
      std::string &label_of_elements,
      int *element_data, int nb_elements,
      int verbose_level);
void conjugate(
```

```
      std::string &label_of_elements,
      std::string &conjugate_data,
      int *element_data, int nb_elements,
      int verbose_level);
void print_action_on_surface(
      std::string &surface_label,
      std::string &label_of_elements,
      int *element_data, int nb_elements,
      int verbose_level);
void element_processing(
      element_processing_description *element_processing_descr,
      int verbose_level);

// any_group_linear.cpp:
void classes_based_on_normal_form(int verbose_level);
void classes(int verbose_level);
void find_singer_cycle(int verbose_level);
void search_element_of_order(int order, int verbose_level);
void find_standard_generators(int order_a,
      int order_b,
      int order_ab,
      int verbose_level);
void isomorphism_Klein_quadric(
      std::string &fname, int verbose_level);
void do_orbits_on_subspaces(
      poset_classification::poset_classification_control *Control,
      orbits_on_subspaces *&OoS,
      int depth, int verbose_level);
void do_tensor_classify(
      std::string &control_label,
      apps_geometry::tensor_classify *&T,
      int depth, int verbose_level);
void do_tensor_permutations(int verbose_level);
void do_linear_codes(
      std::string &control_label,
      int minimum_distance,
      int target_size, int verbose_level);
void do_classify_ovoids(
      apps_geometry::ovoid_classify_description
         *Ovoid_classify_description,
      int verbose_level);
int subspace_orbits_test_set(
      int len, long int *S, int verbose_level);


// any_group_orbits.cpp
void orbits_on_subsets(
```

```
        poset_classification::poset_classification_control *Control,
        poset_classification::poset_classification *&PC,
        int subset_size,
        int verbose_level);
   void orbits_on_poset_post_processing(
        poset_classification::poset_classification *PC,
        int depth,
        int verbose_level);
   void do_orbits_on_group_elements_under_conjugation(
        std::string &fname_group_elements_coded,
        std::string &fname_transporter,
        int verbose_level);

};


// ##############################################################################
// character_table_burnside.cpp
// ##############################################################################

//! character table of a finite group using the algorithm of Burnside


class character_table_burnside {
public:

   void do_it(int n, int verbose_level);
   void create_matrix(
        typed_objects::discreta_matrix &M, int i, int *S, int nb_classes,
      int *character_degree, int *class_size,
      int verbose_level);
   void compute_character_table(
        algebra::a_domain *D, int nb_classes, int *Omega,
      int *character_degree, int *class_size,
      int *&character_table, int verbose_level);
   void compute_character_degrees(
        algebra::a_domain *D,
      int goi, int nb_classes, int *Omega, int *class_size,
      int *&character_degree,
      int verbose_level);
   void compute_omega(
        algebra::a_domain *D, int *N0, int nb_classes,
        int *Mu, int nb_mu, int *&Omega,
        int verbose_level);
   int compute_r0(
        int *N, int nb_classes, int verbose_level);
   void compute_multiplication_constants_center_of_group_ring(
```

```
        actions::action *A,
        induced_actions::action_by_conjugation *ABC,
     groups::schreier *Sch, int nb_classes, int *&N,
     int verbose_level);
  void compute_Distribution_table(
        actions::action *A,
        induced_actions::action_by_conjugation *ABC,
        groups::schreier *Sch, int nb_classes,
     int **Gens, int nb_gens, int t_max, int *&Distribution,
     int verbose_level);
  void multiply_word(
        actions::action *A, int **Gens,
        int *Choice, int t, int *Elt1, int *Elt2,
        int verbose_level);
  void create_generators(
        actions::action *A, int n,
        int **&Elt, int &nb_gens, int f_special,
        int verbose_level);
  void integral_eigenvalues(int *M, int n,
     int *&Lambda,
     int &nb_lambda,
     int *&Mu,
     int *&Mu_mult,
     int &nb_mu,
     int verbose_level);
  void characteristic_poly(
        int *N, int size,
        typed_objects::unipoly &charpoly,
        int verbose_level);
  void double_swap(double &a, double &b);
  int double_Gauss(
        double *A, int m, int n, int *base_cols,
        int verbose_level);
  void double_matrix_print(double *A, int m, int n);
  double double_abs(double x);
  void kernel_columns(
        int n, int nb_base_cols, int *base_cols,
        int *kernel_cols);
  void matrix_get_kernel(
        double *M, int m, int n,
        int *base_cols, int nb_base_cols,
     int &kernel_m, int &kernel_n, double *kernel);
  int double_as_int(double x);

};

// ##############################################################################
```

```
// element_processing_description.cpp
// ##############################################################################

//! describe a process applied to a set of group elements

class element_processing_description {

public:

   int f_input;
   std::string input_label;

   int f_print;

   int f_apply_isomorphism_wedge_product_4to6;

   int f_with_permutation;

   int f_with_fix_structure;

   int f_order_of_products_of_pairs;

   int f_conjugate;
   std::string conjugate_data;

   int f_print_action_on_surface;
   std::string print_action_on_surface_label;



   element_processing_description();
   ~element_processing_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();


};


// ##############################################################################
// group_modification_description.cpp
// ##############################################################################

//! create a new group or group action from an old
```

```
class group_modification_description {

public:

   int f_restricted_action;
   std::string restricted_action_set_text;

   int f_on_k_subspaces;
   int on_k_subspaces_k;

   int f_on_k_subsets;
   int on_k_subsets_k;

   int f_on_wedge_product;

   int f_create_special_subgroup;

   int f_point_stabilizer;
   int point_stabilizer_index;

   int f_projectivity_subgroup;

   int f_subfield_subgroup;
   int subfield_subgroup_index;

   std::vector<std::string> from;

   group_modification_description();
   ~group_modification_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();


};




// ##############################################################################
// group_theoretic_activity_description.cpp
// ##############################################################################


//! description of an activity associated with a group
```

```
class group_theoretic_activity_description {
public:

   int f_apply;
   std::string apply_input;
   std::string apply_element;

   int f_element_processing;
   element_processing_description *element_processing_descr;

   int f_multiply;
   std::string multiply_a;
   std::string multiply_b;

   int f_inverse;
   std::string inverse_a;

   int f_consecutive_powers;
   std::string consecutive_powers_a_text;
   std::string consecutive_powers_exponent_text;

   int f_raise_to_the_power;
   std::string raise_to_the_power_a_text;
   std::string raise_to_the_power_exponent_text;

   int f_export_orbiter;

   int f_export_gap;

   int f_export_magma;

   // GAP:
   int f_canonical_image;
   std::string canonical_image_input_set;

   int f_search_element_of_order;
   int search_element_order;

   int f_find_standard_generators;
   int find_standard_generators_order_a;
   int find_standard_generators_order_b;
   int find_standard_generators_order_ab;

   int f_random_element;
   std::string random_element_label;

   int f_element_rank;
```

```
std::string element_rank_data;

int f_element_unrank;
std::string element_unrank_data;

int f_find_singer_cycle;


int f_classes_based_on_normal_form;

// Magma:
int f_normalizer;

// Magma:
int f_centralizer_of_element;
std::string centralizer_of_element_label;
std::string centralizer_of_element_data;

int f_permutation_representation_of_element;
std::string permutation_representation_element_text;

int f_orbits_on_group_elements_under_conjugation;
std::string orbits_on_group_elements_under_conjugation_fname;
std::string orbits_on_group_elements_under_conjugation_transporter_fname;

// Magma:
int f_normalizer_of_cyclic_subgroup;
std::string normalizer_of_cyclic_subgroup_label;
std::string normalizer_of_cyclic_subgroup_data;

// Magma:
int f_classes;

int f_find_subgroup;
int find_subgroup_order;

int f_report;

   // flags that apply to report:
   int f_report_sylow;
   int f_report_group_table;
   int f_report_classes;


int f_export_group_table;

int f_test_if_geometric;
```

```
int test_if_geometric_depth;

int f_conjugacy_class_of;
std::string conjugacy_class_of_label;
std::string conjugacy_class_of_data;

int f_isomorphism_Klein_quadric;
std::string isomorphism_Klein_quadric_fname;

int f_print_elements;
int f_print_elements_tex;

int f_save_elements_csv;
std::string save_elements_csv_fname;

int f_export_inversion_graphs;
std::string export_inversion_graphs_fname;


int f_multiply_elements_csv_column_major_ordering;
std::string multiply_elements_csv_column_major_ordering_fname1;
std::string multiply_elements_csv_column_major_ordering_fname2;
std::string multiply_elements_csv_column_major_ordering_fname3;

int f_multiply_elements_csv_row_major_ordering;
std::string multiply_elements_csv_row_major_ordering_fname1;
std::string multiply_elements_csv_row_major_ordering_fname2;
std::string multiply_elements_csv_row_major_ordering_fname3;

int f_apply_elements_csv_to_set;
std::string apply_elements_csv_to_set_fname1;
std::string apply_elements_csv_to_set_fname2;
std::string apply_elements_csv_to_set_set;


int f_order_of_products;
std::string order_of_products_elements;

int f_reverse_isomorphism_exterior_square;

int f_is_subgroup_of;
int f_coset_reps;




// orbit stuff:
```

```
   int f_orbit_of;
   int orbit_of_point_idx;

   int f_orbits_on_set_system_from_file;
   std::string orbits_on_set_system_from_file_fname;
   int orbits_on_set_system_first_column;
   int orbits_on_set_system_number_of_columns;

   int f_orbit_of_set_from_file;
   std::string orbit_of_set_from_file_fname;

   // classification of optimal linear codes using poset classification
   int f_linear_codes;
   std::string linear_codes_control;
   int linear_codes_minimum_distance;
   int linear_codes_target_size;




   int f_tensor_permutations;

   int f_classify_ovoids;
   apps_geometry::ovoid_classify_description *Ovoid_classify_description;

   int f_classify_cubic_curves;

   int f_representation_on_polynomials;
   int representation_on_polynomials_degree;



   group_theoretic_activity_description();
   ~group_theoretic_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// group_theoretic_activity.cpp
// ##############################################################################
```

```
//! perform an activity associated with a linear group

class group_theoretic_activity {
public:
   group_theoretic_activity_description *Descr;

   any_group *AG;

   any_group *AG_secondary; // used in is_subgroup_of, coset_reps



   group_theoretic_activity();
   ~group_theoretic_activity();
   void init_group(
         group_theoretic_activity_description *Descr,
         any_group *AG,
         int verbose_level);
   void init_secondary_group(
         group_theoretic_activity_description *Descr,
         any_group *AG_secondary,
         int verbose_level);
   void perform_activity(int verbose_level);

};


// ##############################################################################
// modified_group_create.cpp
// ##############################################################################

//! to create a new group or group action from old ones, using class group_modifi
cation_description

class modified_group_create {

public:
   group_modification_description *Descr;

   std::string label;
   std::string label_tex;

   //strong_generators *initial_strong_gens;

   actions::action *A_base;
```

```
   actions::action *A_previous;
   actions::action *A_modified;

   int f_has_strong_generators;
   groups::strong_generators *Strong_gens;


   modified_group_create();
   ~modified_group_create();
   void modified_group_init(
         group_modification_description *description,
         int verbose_level);
   void create_restricted_action(
         group_modification_description *description,
         int verbose_level);
   void create_action_on_k_subspaces(
         group_modification_description *description,
         int verbose_level);
   void create_action_on_k_subsets(
         group_modification_description *description,
         int verbose_level);
   void create_action_on_wedge_product(
         group_modification_description *description,
         int verbose_level);
   void create_special_subgroup(
         group_modification_description *description,
         int verbose_level);
   void create_point_stabilizer_subgroup(
         group_modification_description *description,
         int verbose_level);
   void create_projectivity_subgroup(
         group_modification_description *description,
         int verbose_level);
   void create_subfield_subgroup(
         group_modification_description *description,
         int verbose_level);

};


// ##############################################################################
// orbit_cascade.cpp
// ##############################################################################

//! a cascade of nested orbit algorithms
```

```
class orbit_cascade {
public:

   int N; // total size of the set
   int k; // size of the subsets

   // we assume that N = 3 * k

   any_group *G;

   poset_classification::poset_classification_control *Control;
   poset_classification::poset_with_group_action *Primary_poset;
   poset_classification::poset_classification *Orbits_on_primary_poset;

   int number_primary_orbits;
   groups::strong_generators **stabilizer_gens;
      // [number_primary_orbits]
   long int *Reps_and_complements;
      // [number_primary_orbits * degree]

   actions::action **A_restricted;
   poset_classification::poset_with_group_action **Secondary_poset;
      // [number_primary_orbits]
   poset_classification::poset_classification **orbits_secondary_poset;
      // [number_primary_orbits]

   int *nb_orbits_secondary;
      // [number_primary_orbits]
   int *flag_orbit_first;
      // [number_primary_orbits]
   int nb_orbits_secondary_total;

   invariant_relations::flag_orbits *Flag_orbits;

   int nb_orbits_reduced;

   invariant_relations::classification_step *Partition_orbits;
      // [nb_orbits_reduced]


   orbit_cascade();
   ~orbit_cascade();
   void init(
         int N, int k, any_group *G,
         std::string &Control_label,
         int verbose_level);
   void downstep(int verbose_level);
```

```
   void upstep(
         std::vector<long int> &Ago, int verbose_level);

};



// ##############################################################################
// orbits_activity_description.cpp
// ##############################################################################


//! description of an action for orbits


class orbits_activity_description {

public:


   int f_report;

   int f_export_something;
   std::string export_something_what;
   int export_something_data1;

   int f_export_trees;

   int f_draw_tree;
   int draw_tree_idx;

   int f_stabilizer;
   int stabilizer_point;

   int f_stabilizer_of_orbit_rep;
   int stabilizer_of_orbit_rep_orbit_idx;

   int f_Kramer_Mesner_matrix;
   int Kramer_Mesner_t;
   int Kramer_Mesner_k;

   int f_recognize;
   std::vector<std::string> recognize;

   int f_report_options;
   poset_classification::poset_classification_report_options
      *report_options;
```

```
   orbits_activity_description();
   ~orbits_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// orbits_activity.cpp
// ##############################################################################


//! perform an activity associated with orbits

class orbits_activity {
public:
   orbits_activity_description *Descr;

   apps_algebra::orbits_create *OC;




   orbits_activity();
   ~orbits_activity();
   void init(
         orbits_activity_description *Descr,
         apps_algebra::orbits_create *OC,
         int verbose_level);
   void perform_activity(int verbose_level);
   void do_report(int verbose_level);
   void do_export(int verbose_level);
   void do_export_trees(int verbose_level);
   void do_draw_tree(int verbose_level);
   void do_stabilizer(int verbose_level);
   void do_stabilizer_of_orbit_rep(int verbose_level);
   void do_Kramer_Mesner_matrix(int verbose_level);
   void do_recognize(int verbose_level);

};
```

```
// ##############################################################################
// orbits_create_description.cpp
// ##############################################################################

//! to describe an orbit problem



class orbits_create_description {

public:

   int f_group;
   std::string group_label;

   int f_on_points;

   int f_on_subsets;
   int on_subsets_size;
   std::string on_subsets_poset_classification_control_label;

   int f_on_subspaces;
   int on_subspaces_dimension;
   std::string on_subspaces_poset_classification_control_label;

   int f_on_tensors;
   int on_tensors_dimension;
   std::string on_tensors_poset_classification_control_label;

   int f_on_partition;
   int on_partition_k;
   std::string on_partition_poset_classification_control_label;

   int f_on_polynomials;
   int on_polynomials_degree;

   int f_classification_by_canonical_form;
   projective_geometry::canonical_form_classifier_description
      *Canonical_form_classifier_description;


#if 0
```

```
   int f_draw_tree;
   int draw_tree_idx;

   int f_recognize;
   std::string recognize_text;
#endif

   orbits_create_description();
   ~orbits_create_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// orbits_create.cpp
// ##############################################################################

//! to create orbits



class orbits_create {

public:
   orbits_create_description *Descr;

   any_group *Group;

   std::string prefix;
   std::string label_txt;
   std::string label_tex;

   int f_has_Orb;
   groups::orbits_on_something *Orb;

   int f_has_On_subsets;
   poset_classification::poset_classification *On_subsets;

   int f_has_On_Subspaces;
   orbits_on_subspaces *On_Subspaces;

   int f_has_On_tensors;
   apps_geometry::tensor_classify *On_tensors;
```

```
   int f_has_Cascade;
   apps_algebra::orbit_cascade *Cascade;

   int f_has_On_polynomials;
   orbits_on_polynomials *On_polynomials;

   int f_has_classification_by_canonical_form;
   projective_geometry::canonical_form_classifier *Canonical_form_classifier;

   orbits_create();
   ~orbits_create();
   void init(
         apps_algebra::orbits_create_description *Descr,
         int verbose_level);

};




// ##############################################################################
// orbits_on_polynomials.cpp
// ##############################################################################


//! orbits of a group on polynomials using Schreier vectors

class orbits_on_polynomials {
public:

   groups::linear_group *LG;
   int degree_of_poly;

   field_theory::finite_field *F;
   actions::action *A;
   int n;
   ring_theory::longinteger_object go;

   ring_theory::homogeneous_polynomial_domain *HPD;

   geometry::projective_space *P;

   actions::action *A2;

   int *Elt1;
   int *Elt2;
   int *Elt3;
```

```
   groups::schreier *Sch;
   ring_theory::longinteger_object full_go;

   std::string fname_base;
   std::string fname_csv;
   std::string fname_report;

   data_structures_groups::orbit_transversal *T;
   int *Nb_pts; // [T->nb_orbits]
   std::vector<std::vector<long int> > Points;


   orbits_on_polynomials();
   ~orbits_on_polynomials();
   void init(
         groups::linear_group *LG,
         int degree_of_poly,
         int verbose_level);
   void compute_points(int verbose_level);
   void report(int verbose_level);
   void report_detailed_list(
         std::ostream &ost,
         int verbose_level);
   void export_something(
         std::string &what, int data1,
         std::string &fname, int verbose_level);
   void export_something_worker(
         std::string &fname_base,
         std::string &what, int data1,
         std::string &fname,
         int verbose_level);


};



// ##############################################################################
// orbits_on_subspaces.cpp
// ##############################################################################


//! orbits of a group on subspaces of a vector space

class orbits_on_subspaces {
public:
```

```
   //group_theoretic_activity *GTA;
   apps_algebra::any_group *Group;

   // local data for orbits on subspaces:
   poset_classification::poset_with_group_action
      *orbits_on_subspaces_Poset;
   poset_classification::poset_classification
      *orbits_on_subspaces_PC;
   algebra::vector_space *orbits_on_subspaces_VS;
   int *orbits_on_subspaces_M;
   int *orbits_on_subspaces_base_cols;


   orbits_on_subspaces();
   ~orbits_on_subspaces();
   void init(
         apps_algebra::any_group *Group,
         poset_classification::poset_classification_control *Control,
         int depth,
         int verbose_level);


};



// ##############################################################################
// polynomial_ring_activity.cpp
// ##############################################################################


//! a polynomial ring activity

class polynomial_ring_activity {
public:

   ring_theory::polynomial_ring_activity_description *Descr;
   ring_theory::homogeneous_polynomial_domain *HPD;



   polynomial_ring_activity();
   ~polynomial_ring_activity();
   void init(
         ring_theory::polynomial_ring_activity_description *Descr,
         ring_theory::homogeneous_polynomial_domain *HPD,
         int verbose_level);
```

```
   void perform_activity(int verbose_level);

};


// ##############################################################################
// vector_ge_builder.cpp
// ##############################################################################



//! to build a vector of group elements based on class vector_ge_description


class vector_ge_builder {

public:

   data_structures_groups::vector_ge_description *Descr;

   data_structures_groups::vector_ge *V;


   vector_ge_builder();
   ~vector_ge_builder();
   void init(
         data_structures_groups::vector_ge_description *Descr,
         int verbose_level);

};









// ##############################################################################
// young.cpp
// ##############################################################################


//! The Young representations of the symmetric group


class young {
```

```
public:
   int n;
   actions::action *A;
   groups::sims *S;
   ring_theory::longinteger_object go;
   int goi;
   int *Elt;
   int *v;

   actions::action *Aconj;
   induced_actions::action_by_conjugation *ABC;
   groups::schreier *Sch;
   groups::strong_generators *SG;
   int nb_classes;
   int *class_size;
   int *class_rep;
   algebra::a_domain *D;

   int l1, l2;
   int *row_parts;
   int *col_parts;
   int *Tableau;

   data_structures::set_of_sets *Row_partition;
   data_structures::set_of_sets *Col_partition;

   data_structures_groups::vector_ge *gens1, *gens2;
   groups::sims *S1, *S2;


   young();
   ~young();
   void init(int n, int verbose_level);
   void create_module(
         int *h_alpha,
      int *&Base, int *&base_cols, int &rk,
      int verbose_level);
   void create_representations(
         int *Base, int *Base_inv, int rk,
      int verbose_level);
   void create_representation(
         int *Base, int *base_cols, int rk,
      int group_elt, int *Mtx, int verbose_level);
      // Mtx[rk * rk * D->size_of_instance_in_int]
   void young_symmetrizer(
         int *row_parts, int nb_row_parts,
      int *tableau,
```

```
      int *elt1, int *elt2, int *elt3,
      int verbose_level);
   void compute_generators(
         int &go1, int &go2, int verbose_level);
   void Maschke(int *Rep,
      int dim_of_module, int dim_of_submodule,
      int *&Mu,
      int verbose_level);
   long int group_ring_element_size(
         actions::action *A, groups::sims *S);
   void group_ring_element_create(
         actions::action *A, groups::sims *S, int *&elt);
   void group_ring_element_free(
         actions::action *A, groups::sims *S, int *elt);
   void group_ring_element_print(
         actions::action *A, groups::sims *S, int *elt);
   void group_ring_element_copy(
         actions::action *A, groups::sims *S,
      int *elt_from, int *elt_to);
   void group_ring_element_zero(
         actions::action *A, groups::sims *S,
      int *elt);
   void group_ring_element_mult(
         actions::action *A, groups::sims *S,
      int *elt1, int *elt2, int *elt3);
};


}}}


#endif /* ORBITER_SRC_LIB_TOP_LEVEL_ALGEBRA_AND_NUMBER_THEORY_TL_ALGEBRA_AND_NUMB
ER_THEORY_H_ */
```

7.3 Coding Theory

```
/*
 * coding_theory_apps.h
 *
 *  Created on: Jul 30, 2022
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_CODING_THEORY_APPS_CODING_THEORY_APPS_H_
#define SRC_LIB_TOP_LEVEL_CODING_THEORY_APPS_CODING_THEORY_APPS_H_



namespace orbiter {
namespace layer5_applications {
namespace apps_coding_theory {



// ##############################################################################
// code_modification_description.cpp
// ##############################################################################

//! unary operators to modify codes


class code_modification_description {

public:

   int f_dual;

   code_modification_description();
   ~code_modification_description();
   int check_and_parse_argument(
      int argc, int &i, std::string *argv,
      int verbose_level);
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();
   void apply(apps_coding_theory::create_code *Code, int verbose_level);

};


// ##############################################################################
```

```
// coding_theoretic_activity_description.cpp
// ##############################################################################

//! description of an activity in coding theory


class coding_theoretic_activity_description {

public:

   int f_report;

   int f_general_code_binary;
   int general_code_binary_n;
   std::string general_code_binary_label;
   std::string general_code_binary_text;

   int f_encode_text_5bits;
   std::string encode_text_5bits_input;
   std::string encode_text_5bits_fname;

   int f_field_induction;
   std::string field_induction_fname_in;
   std::string field_induction_fname_out;
   int field_induction_nb_bits;



   int f_weight_enumerator;
   //std::string weight_enumerator_input_matrix;

   int f_minimum_distance;
   std::string minimum_distance_code_label;

   int f_generator_matrix_cyclic_code;
   int generator_matrix_cyclic_code_n;
   std::string generator_matrix_cyclic_code_poly;

   int f_Sylvester_Hadamard_code;
   int Sylvester_Hadamard_code_n;

   int f_NTT;
   int NTT_n;
   int NTT_q;

   int f_fixed_code;
   std::string fixed_code_perm;
```

```
int f_export_magma;
std::string export_magma_fname;

int f_export_codewords;
std::string export_codewords_fname;

int f_export_codewords_long;
std::string export_codewords_long_fname;

int f_export_codewords_by_weight;
std::string export_codewords_by_weight_fname;

int f_export_genma;
std::string export_genma_fname;

int f_export_checkma;
std::string export_checkma_fname;

int f_make_diagram;

int f_boolean_function_of_code;

int f_embellish;
int embellish_radius;

int f_metric_balls;
int radius_of_metric_ball;

int f_Hamming_space_distance_matrix;
int Hamming_space_distance_matrix_n;

// CRC stuff:
int f_crc32;
std::string crc32_text;

int f_crc32_hexdata;
std::string crc32_hexdata_text;

int f_crc32_test;
int crc32_test_block_length;

int f_crc256_test;
int crc256_test_message_length;
int crc256_test_R;
int crc256_test_k;
```

```
   int f_crc32_remainders;
   int crc32_remainders_message_length;

   int f_crc_encode_file_based;
   std::string crc_encode_file_based_fname_in;
   std::string crc_encode_file_based_fname_out;
   std::string crc_encode_file_based_crc_type;
   int crc_encode_file_based_block_length;

   int f_find_CRC_polynomials;
   int find_CRC_polynomials_nb_errors;
   int find_CRC_polynomials_information_bits;
   int find_CRC_polynomials_check_bits;

   int f_write_code_for_division;
   std::string write_code_for_division_fname;
   std::string write_code_for_division_A;
   std::string write_code_for_division_B;

   int f_polynomial_division_from_file;
   std::string polynomial_division_from_file_fname;
   long int polynomial_division_from_file_r1;

   int f_polynomial_division_from_file_all_k_bit_error_patterns;
   std::string polynomial_division_from_file_all_k_bit_error_patterns_fname;
   int polynomial_division_from_file_all_k_bit_error_patterns_r1;
   int polynomial_division_from_file_all_k_bit_error_patterns_k;



   coding_theoretic_activity_description();
   ~coding_theoretic_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();


};


// ##############################################################################
// coding_theoretic_activity.cpp
// ##############################################################################

//! an activity for codes or finite fields
```

```
class coding_theoretic_activity {

public:

   coding_theoretic_activity_description *Descr;

   int f_has_finite_field;
   field_theory::finite_field *F;

   int f_has_code;
   apps_coding_theory::create_code *Code;


   coding_theoretic_activity();
   ~coding_theoretic_activity();
   void init_field(
         coding_theoretic_activity_description *Descr,
         field_theory::finite_field *F,
         int verbose_level);
   void init_code(
         coding_theoretic_activity_description *Descr,
         create_code *Code,
         int verbose_level);
   void perform_activity(int verbose_level);
   void do_diagram(
         coding_theory::code_diagram *Diagram,
         int verbose_level);


};


// ##############################################################################
// crc_process_description.cpp
// ##############################################################################

//! a description of a crc process


class crc_process_description {

public:

   int f_code;
   std::string code_label;
```

```
   int f_crc_options;
   coding_theory::crc_options_description *Crc_options;



   crc_process_description();
   ~crc_process_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// crc_process.cpp
// ##############################################################################

//! a crc process


class crc_process {

public:

   crc_process_description *Descr;

   create_code *Code;


   int block_length;
   long int information_length;
   long int check_size;

   long int N;
   long int nb_blocks;
   char *buffer;
   char *check_data;

   crc_process();
   ~crc_process();
   void init(crc_process_description *Descr,
         int verbose_level);
   void encode_file(
         std::string &fname_in, std::string &fname_out,
```

```
        int verbose_level);
   void encode_block(
        long int L,
        int verbose_level);

};



// ##############################################################################
// create_code_description.cpp
// ##############################################################################

//! a description of a code using command line arguments


class create_code_description {

public:

   int f_field;
   std::string field_label;

   int f_generator_matrix;
   std::string generator_matrix_label_genma;

   int f_basis;
   int basis_n;
   std::string basis_label;

   int f_long_code;
   int long_code_n;
   std::vector<std::string> long_code_generators;

   int f_projective_set;
   int projective_set_nmk;
   std::string projective_set_set;

   int f_columns_of_generator_matrix;
   int columns_of_generator_matrix_k;
   std::string columns_of_generator_matrix_set;

   int f_Reed_Muller;
   int Reed_Muller_m;

   int f_BCH;
   int BCH_n;
```

```
   int BCH_d;

   int f_Reed_Solomon;
   int Reed_Solomon_n;
   int Reed_Solomon_d;

   int f_Gilbert_Varshamov;
   int Gilbert_Varshamov_n;
   int Gilbert_Varshamov_k;
   int Gilbert_Varshamov_d;

   int f_ttpA;
   std::string ttpA_field_label;

   int f_ttpB;
   std::string ttpB_field_label;

   std::vector<code_modification_description> Modifications;


   create_code_description();
   ~create_code_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();


};

// ##############################################################################
// create_code.cpp
// ##############################################################################


//! creates a code from a description with create_code_description


class create_code {
public:

   create_code_description *description;

   std::string label_txt;
   std::string label_tex;

   int f_field;
```

```
field_theory::finite_field *F;

int f_has_generator_matrix;
int *genma; // [k * n]

int f_has_check_matrix;
int *checkma; // [nmk * n]
int n;
int k;
int nmk;
int d;

coding_theory::create_BCH_code *Create_BCH_code; // if BCH code
coding_theory::create_RS_code *Create_RS_code; // if RS code


create_code();
~create_code();
void init(
      create_code_description *description,
      int verbose_level);
void dual_code(int verbose_level);
void export_magma(
      std::string &fname, int verbose_level);
void create_genma_from_checkma(int verbose_level);
void create_checkma_from_genma(int verbose_level);
void export_codewords(
      std::string &fname, int verbose_level);
void export_codewords_long(
      std::string &fname, int verbose_level);
void export_codewords_by_weight(
      std::string &fname_base, int verbose_level);
void export_genma(
      std::string &fname, int verbose_level);
void export_checkma(
      std::string &fname, int verbose_level);
void weight_enumerator(int verbose_level);
void fixed_code(
   long int *perm, int n,
   int verbose_level);
void make_diagram(
      int f_embellish, int embellish_radius,
      int f_metric_balls, int radius_of_metric_ball,
      coding_theory::code_diagram *&Diagram,
      int verbose_level);
void polynomial_representation_of_boolean_function(
      int verbose_level);
```

```
   void report(int verbose_level);
   void report2(std::ofstream &ost, int verbose_level);

};



}}}



#endif /* SRC_LIB_TOP_LEVEL_CODING_THEORY_APPS_CODING_THEORY_APPS_H_ */
```

7.4 Combinatorics

```
/*
 * tl_combinatorics.h
 *
 *  Created on: Oct 27, 2019
 *      Author: betten
 */

#ifndef ORBITER_SRC_LIB_TOP_LEVEL_COMBINATORICS_TL_COMBINATORICS_H_
#define ORBITER_SRC_LIB_TOP_LEVEL_COMBINATORICS_TL_COMBINATORICS_H_


namespace orbiter {
namespace layer5_applications {
namespace apps_combinatorics {


// ##############################################################################
// boolean_function_classify.cpp
// ##############################################################################


//! classification of boolean functions


class boolean_function_classify {

public:

   combinatorics::boolean_function_domain *BF;

   // group stuff:
   actions::action *A;
   //data_structures_groups::vector_ge *nice_gens;

   induced_actions::action_on_homogeneous_polynomials *AonHPD;
   groups::strong_generators *SG;
   ring_theory::longinteger_object go;

   actions::action *A_affine;
      // restricted action on affine points

   boolean_function_classify();
   ~boolean_function_classify();

   void init_group(
         combinatorics::boolean_function_domain *BF,
```

```
        actions::action *A,
        int verbose_level);
   void search_for_bent_functions(int verbose_level);

};









// ##############################################################################
// combinatorial_object_activity_description.cpp
// ##############################################################################


//! description of an activity for a combinatorial object

class combinatorial_object_activity_description {
public:


   // options that apply to GOC = geometric_object_create

   int f_save;

   int f_save_as;
   std::string save_as_fname;

   int f_extract_subset;
   std::string extract_subset_set;
   std::string extract_subset_fname;

   int f_line_type;
   std::string line_type_projective_space_label;
   std::string line_type_prefix;

   int f_conic_type;
   int conic_type_threshold;

   int f_non_conical_type;

   int f_ideal;
   std::string ideal_ring_label;
```

```
   // options that apply to IS = data_input_stream

   int f_canonical_form_PG;
   std::string canonical_form_PG_PG_label;
   int f_canonical_form_PG_has_PA;
   projective_geometry::projective_space_with_action
      *Canonical_form_PG_PA;
   combinatorics::classification_of_objects_description
      *Canonical_form_PG_Descr;

   int f_canonical_form;
   combinatorics::classification_of_objects_description
      *Canonical_form_Descr;

   int f_report;
   combinatorics::classification_of_objects_report_options
      *Classification_of_objects_report_options;

   int f_draw_incidence_matrices;
   std::string draw_incidence_matrices_prefix;

   int f_test_distinguishing_property;
   std::string test_distinguishing_property_graph;

   int f_unpack_from_restricted_action;
   std::string unpack_from_restricted_action_prefix;
   std::string unpack_from_restricted_action_group_label;

   int f_line_covering_type;
   std::string line_covering_type_prefix;
   std::string line_covering_type_projective_space;
   std::string line_covering_type_lines;


   combinatorial_object_activity_description();
   ~combinatorial_object_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};

// ##############################################################################
// combinatorial_object_activity.cpp
```

```
// ##############################################################################


//! perform an activity for a combinatorial object

class combinatorial_object_activity {
public:
   combinatorial_object_activity_description *Descr;

   int f_has_geometric_object;
   geometry::geometric_object_create *GOC;

   int f_has_input_stream;
   data_structures::data_input_stream *IS;


   combinatorial_object_activity();
   ~combinatorial_object_activity();
   void init(
         combinatorial_object_activity_description *Descr,
         geometry::geometric_object_create *GOC,
         int verbose_level);
   void init_input_stream(
         combinatorial_object_activity_description *Descr,
         data_structures::data_input_stream *IS,
         int verbose_level);
   void perform_activity(int verbose_level);
   void perform_activity_geometric_object(int verbose_level);
   void perform_activity_input_stream(int verbose_level);
   void do_save(
         std::string &save_as_fname,
         int f_extract,
         long int *extract_idx_set, int extract_size,
         int verbose_level);
   void post_process_classification(
         combinatorics::classification_of_objects *CO,
         object_with_properties *&OwP,
         int f_projective_space,
         projective_geometry::projective_space_with_action *PA,
         std::string &prefix,
         int verbose_level);
   void classification_report(
         combinatorics::classification_of_objects *CO,
         object_with_properties *OwP, int verbose_level);
   void latex_report(
         combinatorics::classification_of_objects_report_options
            *Report_options,
```

```
        combinatorics::classification_of_objects *CO,
        object_with_properties *OwP,
        int verbose_level);
  void report_all_isomorphism_types(
        std::ostream &fp,
        combinatorics::classification_of_objects_report_options
           *Report_options,
        combinatorics::classification_of_objects *CO,
        object_with_properties *OwP,
        int verbose_level);
  void report_isomorphism_type(
        std::ostream &fp,
        combinatorics::classification_of_objects_report_options
           *Report_options,
        combinatorics::classification_of_objects *CO,
        object_with_properties *OwP,
        int i, int verbose_level);
  void report_object(
        std::ostream &fp,
        combinatorics::classification_of_objects_report_options
           *Report_options,
        combinatorics::classification_of_objects *CO,
        object_with_properties *OwP,
        int object_idx,
        int verbose_level);
  void draw_incidence_matrices(
        std::string &prefix,
        data_structures::data_input_stream *IS,
        int verbose_level);
  void unpack_from_restricted_action(
        std::string &prefix,
        std::string &group_label,
        data_structures::data_input_stream *IS,
        int verbose_level);
  void line_covering_type(
        std::string &prefix,
        std::string &projective_space_label,
        std::string &lines,
        data_structures::data_input_stream *IS,
        int verbose_level);
  void line_type(
        std::string &prefix,
        std::string &projective_space_label,
        data_structures::data_input_stream *IS,
        int verbose_level);

};
```

```
// ##############################################################################
// combinatorics_global.cpp
// ##############################################################################


//! combinatorics stuff


class combinatorics_global {

public:

   combinatorics_global();
   ~combinatorics_global();
   void create_design_table(
         design_create *DC,
         std::string &problem_label,
         design_tables *&T,
         groups::strong_generators *Gens,
         int verbose_level);
   void load_design_table(design_create *DC,
         std::string &problem_label,
         design_tables *&T,
         groups::strong_generators *Gens,
         int verbose_level);

   void Hill_cap56(
      char *fname, int &nb_Pts, long int *&Pts,
      int verbose_level);
   void append_orbit_and_adjust_size(
         groups::schreier *Orb,
         int idx, int *set, int &sz);

};


// ##############################################################################
// delandtsheer_doyen_description.cpp
// ##############################################################################

#define MAX_MASK_TESTS 1000
```

```
//! description of the problem for delandtsheer_doyen


class delandtsheer_doyen_description {
public:

   int f_depth;
   int depth;

   int f_d1;
   int d1;

   int f_d2;
   int d2;

   int f_q1;
   int q1;

   int f_q2;
   int q2;

   int f_group_label;
   std::string group_label;

   int f_mask_label;
   std::string mask_label;

   int f_problem_label;
   std::string problem_label;



   int DELANDTSHEER_DOYEN_X;
   int DELANDTSHEER_DOYEN_Y;
   int f_K;
   int K;

   int f_pair_search_control;
   poset_classification::poset_classification_control
      *Pair_search_control;

   int f_search_control;
   poset_classification::poset_classification_control
      *Search_control;

   // row intersection type
```

```
   int f_R;
   int nb_row_types;
   int *row_type; // [nb_row_types + 1]

   // col intersection type
   int f_C;
   int nb_col_types;
   int *col_type; // [nb_col_types + 1]


   int f_nb_orbits_on_blocks;
   int nb_orbits_on_blocks;


   // mask related test:
   int nb_mask_tests;
   int mask_test_level[MAX_MASK_TESTS];
   int mask_test_who[MAX_MASK_TESTS];
      // 1 = x
      // 2 = y
      // 3 = x+y
      // 4 = singletons
   int mask_test_what[MAX_MASK_TESTS];
      // 1 = eq
      // 2 = ge
      // 3 = le
   int mask_test_value[MAX_MASK_TESTS];

   int f_singletons;
   int f_subgroup;
   std::string subgroup_gens;
   std::string subgroup_order;

   int f_search_wrt_subgroup;


   delandtsheer_doyen_description();
   ~delandtsheer_doyen_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};
```

```
// ##############################################################################
// delandtsheer_doyen.cpp
// ##############################################################################



//! search for line transitive point imprimitive linear spaces as described by De
landtsheer and Doyen



class delandtsheer_doyen {
public:

   delandtsheer_doyen_description *Descr;

   field_theory::finite_field *F1;
   field_theory::finite_field *F2;

   int Xsize; // = D = q1 = # of rows
   int Ysize; // = C = q2 = # of cols

   int V; // = Xsize * Ysize
   int b;
   long int *line;        // [K];
   int *row_sum; // [Xsize]
   int *col_sum; // [Ysize]


   groups::matrix_group *M1;
   groups::matrix_group *M2;
   actions::action *A1;
   actions::action *A2;

   actions::action *A;
   actions::action *A0;

   groups::strong_generators *SG;
   ring_theory::longinteger_object go;
   groups::direct_product *P;
   poset_classification::poset_with_group_action *Poset_pairs;
   poset_classification::poset_with_group_action *Poset_search;
   poset_classification::poset_classification *Pairs;
   poset_classification::poset_classification *Gen;

   // orbits on pairs:
   int *pair_orbit; // [V * V]
```

```
int nb_orbits;
int *transporter;
int *tmp_Elt;
int *orbit_length; // [nb_orbits]
int *orbit_covered; // [nb_orbits]
int *orbit_covered_max; // [nb_orbits]
   // orbit_covered_max[i] = orbit_length[i] / b;
int *orbits_covered; // [K * K]


// intersection type tests:

int inner_pairs_in_rows;
int inner_pairs_in_cols;

// row intersection type
int *row_type_cur; // [nb_row_types + 1]
int *row_type_this_or_bigger; // [nb_row_types + 1]

// col intersection type
int *col_type_cur; // [nb_col_types + 1]
int *col_type_this_or_bigger; // [nb_col_types + 1]



// for testing the mask:
int *f_row_used; // [Xsize];
int *f_col_used; // [Ysize];
int *row_idx; // [Xsize];
int *col_idx; // [Ysize];
int *singletons; // [K];

// temporary data
int *row_col_idx; // [Xsize];
int *col_row_idx; // [Ysize];

long int *live_points; // [V]
int nb_live_points;

delandtsheer_doyen();
~delandtsheer_doyen();
void init(delandtsheer_doyen_description *Descr,
      int verbose_level);
void show_generators(int verbose_level);
void search_singletons(int verbose_level);
void search_starter(int verbose_level);
void compute_orbits_on_pairs(
```

```
        groups::strong_generators *Strong_gens,
        int verbose_level);
   groups::strong_generators *scan_subgroup_generators(
        int verbose_level);
   void create_monomial_group(int verbose_level);
   void create_action(int verbose_level);
   void create_graph(long int *line0, int len, int verbose_level);
   int find_pair_orbit(int i, int j, int verbose_level);
   int find_pair_orbit_by_tracing(int i, int j, int verbose_level);
   void compute_pair_orbit_table(int verbose_level);
   void write_pair_orbit_file(int verbose_level);
   void print_mask_test_i(std::ostream &ost, int i);
   void early_test_func(
        long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   int check_conditions(long int *S, int len, int verbose_level);
   int check_orbit_covering(long int *line, int len, int verbose_level);
   int check_row_sums(long int *line, int len, int verbose_level);
   int check_col_sums(long int *line, int len, int verbose_level);
   int check_mask(long int *line, int len, int verbose_level);
   void get_mask_core_and_singletons(
      long int *line, int len,
      int &nb_rows_used, int &nb_cols_used,
      int &nb_singletons, int verbose_level);
};




// ##############################################################################
// design_activity_description.cpp
// ##############################################################################

//! to describe an activity for a design



class design_activity_description {

public:


   int f_load_table;
   std::string load_table_label;
   std::string load_table_group;
```

```
   std::string load_table_H_label;
   std::string load_table_H_group_order;
   std::string load_table_H_gens;
   int load_table_selected_orbit_length;


   int f_canonical_form;
   combinatorics::classification_of_objects_description
      *Canonical_form_Descr;

   int f_extract_solutions_by_index_csv;
   int f_extract_solutions_by_index_txt;
   std::string extract_solutions_by_index_label;
   std::string extract_solutions_by_index_group;
   std::string extract_solutions_by_index_fname_solutions_in;
   std::string extract_solutions_by_index_fname_solutions_out;
   std::string extract_solutions_by_index_prefix;

   int f_export_inc;
   int f_intersection_matrix;
   int f_export_blocks;
   int f_row_sums;
   int f_tactical_decomposition;

   design_activity_description();
   ~design_activity_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// design_activity.cpp
// ##############################################################################

//! an activity for a design



class design_activity {

public:
   design_activity_description *Descr;
```

```
design_activity();
~design_activity();
void perform_activity(
      design_activity_description *Descr,
      design_create *DC, int verbose_level);
void do_extract_solutions_by_index(
      design_create *DC,
      std::string &label,
      std::string &group_label,
      std::string &fname_in,
      std::string &fname_out,
      std::string &prefix_text,
      int f_csv_format,
      int verbose_level);
void do_create_table(
      design_create *DC,
      std::string &label,
      std::string &group_label,
      int verbose_level);
void do_load_table(
      design_create *DC,
      std::string &label,
      std::string &group_label,
      std::string &H_label,
      std::string &H_go_text,
      std::string &H_generators_data,
      int selected_orbit_length,
      int verbose_level);
void do_canonical_form(
      combinatorics::classification_of_objects_description
         *Canonical_form_Descr,
      int verbose_level);
void do_export_inc(
      design_create *DC,
      int verbose_level);
void do_intersection_matrix(
      design_create *DC,
      int verbose_level);
void do_export_blocks(
      design_create *DC,
      int verbose_level);
void do_row_sums(
      design_create *DC,
      int verbose_level);
void do_tactical_decomposition(
```

```
        design_create *DC,
        int verbose_level);

};



// ##############################################################################
// design_create_description.cpp
// ##############################################################################

//! to describe the construction of a known design from the command line



class design_create_description {

public:

   int f_field;
   std::string field_label;

   int f_catalogue;
   int iso;

   int f_family;
   std::string family_name;

   int f_list_of_blocks_coded;
   int list_of_blocks_coded_v;
   int list_of_blocks_coded_k;
   std::string list_of_blocks_coded_label;

   int f_list_of_sets_coded;
   int list_of_sets_coded_v;
   std::string list_of_sets_coded_label;

   int f_list_of_blocks_coded_from_file;
   std::string list_of_blocks_coded_from_file_fname;

   int f_list_of_blocks_from_file;
   int list_of_blocks_from_file_v;
   std::string list_of_blocks_from_file_fname;

   int f_wreath_product_designs;
   int wreath_product_designs_n;
   int wreath_product_designs_k;
```

```
   int f_no_group;


   design_create_description();
   ~design_create_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// design_create.cpp
// ##############################################################################

//! to create a known design using a description from class design_create_descrip
tion



class design_create {

public:
   design_create_description *Descr;

   std::string prefix;
   std::string label_txt;
   std::string label_tex;

   int q;
   field_theory::finite_field *F;
   int k;

   actions::action *A;
      // Sym(degree)
   actions::action *A2;
      // Sym(degree), in the action on k-subsets


   actions::action *Aut;
   // PGGL(3,q) in case of PG_2_q with q not prime
   // PGL(3,q) in case of PG_2_q with q prime
   actions::action *Aut_on_lines; // Aut induced on lines
```

```
   int degree;

   int f_has_set;
   long int *set;
   int sz; // = b, the number of blocks

   int f_has_group;
   groups::strong_generators *Sg;


   projective_geometry::projective_space_with_action *PA;
   geometry::projective_space *P;

   int *block; // [k]

   int v;
   int b;
   int nb_inc;
   int f_has_incma;
   int *incma; // [v * b]

   design_create();
   ~design_create();
   void init(
         apps_combinatorics::design_create_description *Descr,
         int verbose_level);
   void create_design_PG_2_q(field_theory::finite_field *F,
         long int *&set, int &sz, int &k, int verbose_level);
   void unrank_block_in_PG_2_q(int *block,
         int rk, int verbose_level);
   int rank_block_in_PG_2_q(int *block,
         int verbose_level);
   int get_nb_colors_as_two_design(int verbose_level);
   int get_color_as_two_design_assume_sorted(
         long int *design, int verbose_level);
   void compute_incidence_matrix(int verbose_level);
   void compute_incidence_matrix_from_blocks(
         int *blocks, int nb_blocks, int k, int verbose_level);

};


// ##############################################################################
// design_tables.cpp
// ##############################################################################

//! a set of designs to be used for a large set
```

```
class design_tables {

public:

   actions::action *A;
   actions::action *A2;
   long int *initial_set;
   int design_size;
   std::string label;
   std::string fname_design_table;
   groups::strong_generators *Strong_generators;

   int nb_designs;
   long int *the_table; // [nb_designs * design_size]


   design_tables();
   ~design_tables();
   void init(actions::action *A,
         actions::action *A2,
         long int *initial_set, int design_size,
         std::string &label,
         groups::strong_generators *Strong_generators,
         int verbose_level);
   void create_table(int verbose_level);
   void create_action(
         actions::action *&A_on_designs,
         int verbose_level);
   void extract_solutions_by_index(
         int nb_sol, int Index_width, int *Index,
         std::string &ouput_fname_csv,
         int verbose_level);
   void make_reduced_design_table(
         long int *set, int set_sz,
         long int *&reduced_table,
         long int *&reduced_table_idx,
         int &nb_reduced_designs,
         int verbose_level);
   void init_from_file(
         actions::action *A,
         actions::action *A2,
         long int *initial_set, int design_size,
         std::string &label,
         groups::strong_generators *Strong_generators,
         int verbose_level);
```

```
   int test_if_table_exists(
         std::string &label,
         int verbose_level);
   void save(int verbose_level);
   void load(int verbose_level);
   int test_if_designs_are_disjoint(int i, int j);
   int test_set_within_itself(
         long int *set_of_designs_by_index,
         int set_size);
   int test_between_two_sets(
         long int *set_of_designs_by_index1, int set_size1,
         long int *set_of_designs_by_index2, int set_size2);

};




// ##############################################################################
// difference_set_in_heisenberg_group.cpp
// ##############################################################################


//! to find difference sets in Heisenberg groups following Tao


class difference_set_in_heisenberg_group {

public:
   std::string fname_base;

   int n;
   int q;
   field_theory::finite_field *F;
   algebra::heisenberg *H;
   int *Table;
   int *Table_abv;
   int *gens;
   int nb_gens;

   actions::action *A;
   groups::strong_generators *Aut_gens;
   ring_theory::longinteger_object Aut_order;

   int given_base_length; // = nb_gens
   long int *given_base; // = gens
   int *base_image;
```

```
int *base_image_elts;
int *E1;
int rk_E1;

std::string prefix;
std::string fname_magma_out;
groups::sims *Aut;
groups::sims *U;
ring_theory::longinteger_object U_go;
data_structures_groups::vector_ge *U_gens;
groups::schreier *Sch;


// N = normalizer of U in Aut
int *N_gens;
int N_nb_gens, N_go;
actions::action *N;
ring_theory::longinteger_object N_order;

actions::action *N_on_orbits;
int *Paired_with;
int nb_paired_orbits;
long int *Pairs;
int *Pair_orbit_length;


int *Pairs_of_type1;
int nb_pairs_of_type1;
int *Pairs_of_type2;
int nb_pairs_of_type2;
int *Sets1;
int *Sets2;


long int *Short_pairs;
long int *Long_pairs;

int *f_orbit_select;
int *Short_orbit_inverse;

actions::action *A_on_short_orbits;
int nb_short_orbits;
int nb_long_orbits;

poset_classification::poset_classification *gen;
```

```
   void init(int n,
         field_theory::finite_field *F, int verbose_level);
   void do_n2q3(int verbose_level);
   void check_overgroups_of_order_nine(int verbose_level);
   void create_minimal_overgroups(int verbose_level);
   void early_test_func(long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);

};



// ##############################################################################
// flag_orbits_incidence_structure.cpp
// ##############################################################################

//! classification of flag orbits of an incidence structure




class flag_orbits_incidence_structure {

public:

   object_with_properties *OwP;

   int nb_rows;
   int nb_cols;

   int f_flag_orbits_have_been_computed;
   int nb_flags;
   int *Flags; // [nb_flags]
   long int *Flag_table; // [nb_flags * 2]

   actions::action *A_on_flags;

   groups::orbits_on_something *Orb;

   flag_orbits_incidence_structure();
   ~flag_orbits_incidence_structure();
   void init(object_with_properties *OwP,
         int f_anti_flags, actions::action *A_perm,
```

```
         groups::strong_generators *SG, int verbose_level);
   int find_flag(int i, int j);
   void report(std::ostream &ost, int verbose_level);

};




// ##############################################################################
// hadamard_classify.cpp
// ##############################################################################

//! classification of Hadamard matrices




class hadamard_classify {

public:
   int n;
   int N, N2;
   data_structures::bitvector *Bitvec;
   graph_theory::colored_graph *CG;

   actions::action *A;

   int *v;

   poset_classification::poset_classification *gen;
   int nb_orbits;

   void init(int n,
         int f_draw, int verbose_level,
         int verbose_level_clique);
   int clique_test(long int *set, int sz);
   void early_test_func(long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   int dot_product(int a, int b, int n);
};
```

```
// ##############################################################################
// hall_system_classify.cpp
// ##############################################################################

//! classification of Hall systems


class hall_system_classify {
public:
   //int e;
   int n; // 3^e
   int nm1; // n-1
      // number of points different from the reflection point
   int nb_pairs;
      // nm1 / 2
      // = number of lines (=triples) through the reflection point
      // = number of lines (=triples) through any point
   int nb_pairs2; // = nm1 choose 2
      // number of pairs of points
      // different from the reflection point.
      // these are the pairs of points that are covered by the
      // triples that we will choose.
      // The other pairs have been covered by the lines through
      // the reflection point,
      // so they are fine because we assume that
      // these lines exist.
   int nb_blocks_overall; // {n \choose 2} / 6
   int nb_blocks_needed; // nb_blocks_overall - (n - 1) / 2
   int nb_orbits_needed; // nb_blocks_needed / 2
   int depth;
   int N;
      // {nb_pairs choose 3} * 8
      // {nb_pairs choose 3}
      // counts the number of ways to choose three lines
      // through the reflection point.
      // the times 8 is because every triple of lines through the
      // reflection point has 2^3 ways
      // of choosing one point on each line.
   int N0; // {nb_pairs choose 3} * 4
   int *row_sum; // [nm1]
      // this is where we test whether each of the
      // points different from the reflection point lies on
      // the right number of triples.
      // The right number is nb_pairs
   int *pair_covering; // [nb_pairs2]
      // this is where we test whether each of the
```

```
   // pairs of points not including the reflection point
   // is covered once


long int *triples; // [N0 * 6]
   // a table of all triples so that
   // we can induce the group action on to them.


actions::action *A;
   // The symmetric group on nm1 points.
actions::action *A_on_triples;
   // the induced action on unordered triples
   // as stored in triples[].
groups::strong_generators *Strong_gens_Hall_reflection;
   // the involution which switches the
   // points on every line through
   // the center (other than the center).
groups::strong_generators *Strong_gens_normalizer;
   // Strong generators for the normalizer
   // of the involution.
groups::sims *S;
   // The normalizer of the involution

std::string prefix;
std::string fname_orbits_on_triples;
groups::schreier *Orbits_on_triples;
   // Orbits of the reflection group on triples.
actions::action *A_on_orbits;
   // Induced action of A_on_triples
   // on the orbit of the reflection group.
int f_play_it_safe;

poset_classification::poset_classification_control *Control;
poset_classification::poset_with_group_action *Poset;
   // subset lattice for action A_on_orbits
poset_classification::poset_classification *PC;
   // Classification of subsets in the action A_on_orbits


hall_system_classify();
~hall_system_classify();
void init(int argc, const char **argv,
      int n, int depth,
      int verbose_level);
void orbits_on_triples(int verbose_level);
void print(std::ostream &ost, long int *S, int len);
```

```
   void unrank_triple(long int *T, int rk);
   void unrank_triple_pair(
         long int *T1, long int *T2, int rk);
   void early_test_func(long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
};









// ##############################################################################
// large_set_activity_description.cpp
// ##############################################################################

//! description of an activity for a spread table


class large_set_activity_description {
public:



   large_set_activity_description();
   ~large_set_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);

};


// ##############################################################################
// large_set_activity.cpp
// ##############################################################################

//! an activity for a spread table


class large_set_activity {
public:

   large_set_activity_description *Descr;
```

```
   large_set_was *LSW;



   large_set_activity();
   ~large_set_activity();
   void perform_activity(
         large_set_activity_description *Descr,
         large_set_was *LSW, int verbose_level);

};



// ##############################################################################
// large_set_classify.cpp
// ##############################################################################

//! classification of large sets of designs

class large_set_classify {
public:
   design_create *DC;
   int design_size;
      // = DC->sz = b,
      // the number of blocks in the design
   int nb_points; // = DC->A->degree
   int nb_lines; // = DC->A2->degree
   int search_depth;

   std::string problem_label;

   int f_lexorder_test;
   int size_of_large_set; // = nb_lines / design_size


   design_tables *Design_table;

   int nb_colors;
      // = DC->get_nb_colors_as_two_design
   int *design_color_table; // [nb_designs]

   actions::action *A_on_designs;


   data_structures::bitvector *Bitvec;
   int *degree;
```

```
   poset_classification::poset_classification_control *Control;
   poset_classification::poset_with_group_action *Poset;
   poset_classification::poset_classification *gen;

   int nb_needed;




   large_set_classify();
   ~large_set_classify();
   void init(design_create *DC,
         design_tables *T,
         int verbose_level);
   void create_action_and_poset(int verbose_level);
   void compute(int verbose_level);
   void read_classification(
         data_structures_groups::orbit_transversal *&T,
         int level, int verbose_level);
   void read_classification_single_case(
         data_structures_groups::set_and_stabilizer *&Rep,
         int level, int case_nr, int verbose_level);
   void compute_colors(
         design_tables *Design_table, int *&design_color_table,
         int verbose_level);
   int test_if_designs_are_disjoint(int i, int j);

};









// ##############################################################################
// large_set_was_activity_description.cpp
// ##############################################################################

//! description of an activity for a large set search with assumed symmetry


class large_set_was_activity_description {
public:
```

```
   int f_normalizer_on_orbits_of_a_given_length;
   int normalizer_on_orbits_of_a_given_length_length;
   int normalizer_on_orbits_of_a_given_length_nb_orbits;
   poset_classification::poset_classification_control
      *normalizer_on_orbits_of_a_given_length_control;

   int f_create_graph_on_orbits_of_length;
   std::string create_graph_on_orbits_of_length_fname;
   int create_graph_on_orbits_of_length_length;

   int f_create_graph_on_orbits_of_length_based_on_N_orbits;
   std::string create_graph_on_orbits_of_length_based_on_N_orbits_fname_mask;
   int create_graph_on_orbits_of_length_based_on_N_orbits_length;
   int create_graph_on_orbits_of_length_based_on_N_nb_N_orbits_preselected;
   int create_graph_on_orbits_of_length_based_on_N_orbits_r;
   int create_graph_on_orbits_of_length_based_on_N_orbits_m;

   int f_read_solution_file;
   int read_solution_file_orbit_length;
   std::string read_solution_file_name;



   large_set_was_activity_description();
   ~large_set_was_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// large_set_was_activity.cpp
// ##############################################################################

//! an activity for a large set search with assumed symmetry


class large_set_was_activity {
public:

   large_set_was_activity_description *Descr;
   large_set_was *LSW;
```

```
   large_set_was_activity();
   ~large_set_was_activity();
   void perform_activity(
         large_set_was_activity_description *Descr,
         large_set_was *LSW, int verbose_level);
   void do_normalizer_on_orbits_of_a_given_length(
         int select_orbits_of_length_length, int verbose_level);

};




// ##############################################################################
// large_set_was_description.cpp
// ##############################################################################

//! command line description of tasks for large sets with assumed symmetry

class large_set_was_description {
public:


   int f_H;
   std::string H_go;
   std::string H_generators_text;

   int f_N;
   std::string N_go;
   std::string N_generators_text;

   int f_report;

   int f_prefix;
   std::string prefix;

   int f_selected_orbit_length;
   int selected_orbit_length;

   large_set_was_description();
   ~large_set_was_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};
```

```
// ##############################################################################
// large_set_was.cpp
// ##############################################################################

//! classification of large sets of designs with assumed symmetry

class large_set_was {
public:

   large_set_was_description *Descr;

   large_set_classify *LS;

   groups::strong_generators *H_gens;

   groups::orbits_on_something *H_orbits;


   groups::strong_generators *N_gens;

   groups::orbits_on_something *N_orbits;


   // used in do_normalizer_on_orbits_of_a_given_length:
   int orbit_length;
   int nb_of_orbits_to_choose;
   int type_idx;
      // orbits of length orbit_length
      // in H_orbits->Orbits_classified
   long int *Orbit1;
   long int *Orbit2;

   actions::action *A_on_orbits;
      // action on H_orbits->Sch
   actions::action *A_on_orbits_restricted;
      // action A_on_orbits restricted to
      // H_orbits->Orbits_classified->Sets[type_idx]


   // used in do_normalizer_on_orbits_of_
   // a_given_length_multiple_orbits
   poset_classification::poset_classification *PC;
   poset_classification::poset_classification_control *Control;
   poset_classification::poset_with_group_action *Poset;
```

```
int orbit_length2;
int type_idx2;
   // orbits of length orbit_length2
   // in H_orbits->Orbits_classified

int selected_type_idx;


large_set_was();
~large_set_was();
void init(large_set_was_description *Descr,
      large_set_classify *LS,
      int verbose_level);
void do_normalizer_on_orbits_of_a_given_length(
      int orbit_length,
      int nb_of_orbits_to_choose,
      poset_classification::poset_classification_control *Control,
      int verbose_level);
void do_normalizer_on_orbits_of_a_given_length_single_orbit(
      int orbit_length,
      int verbose_level);
void do_normalizer_on_orbits_of_a_given_length_multiple_orbits(
      int orbit_length,
      int nb_of_orbits_to_choose,
      poset_classification::poset_classification_control *Control,
      int verbose_level);
void create_graph_on_orbits_of_length(
      std::string &fname, int orbit_length,
      int verbose_level);
void create_graph_on_orbits_of_length_based_on_N_orbits(
      std::string &fname_mask,
      int orbit_length2, int nb_N_orbits_preselected,
      int orbit_r, int orbit_m,
      int verbose_level);
void read_solution_file(
      std::string &solution_file_name,
      long int *starter_set,
      int starter_set_sz,
      int orbit_length,
      int verbose_level);
void normalizer_orbits_early_test_func(long int *S, int len,
   long int *candidates, int nb_candidates,
   long int *good_candidates, int &nb_good_candidates,
   int verbose_level);
int normalizer_orbits_check_conditions(
```

```
         long int *S, int len, int verbose_level);

};

void large_set_was_normalizer_orbits_early_test_func_callback(
      long int *S, int len,
   long int *candidates, int nb_candidates,
   long int *good_candidates, int &nb_good_candidates,
   void *data, int verbose_level);
int large_set_was_design_test_orbit(
      long int *orbit, int orbit_length,
      void *extra_data);
int large_set_was_classify_test_pair_of_orbits(
      long int *orbit1, int orbit_length1,
      long int *orbit2, int orbit_length2,
      void *extra_data);



// ##############################################################################
// object_with_properties.cpp
// ##############################################################################

//! object properties which are derived from nauty canonical form


class object_with_properties {
public:

   geometry::object_with_canonical_form *OwCF;

   std::string label;

   data_structures::nauty_output *NO;

   int f_projective_space;
   projective_geometry::projective_space_with_action *PA;
   groups::strong_generators *SG;
      // only used if f_projective_space

   actions::action *A_perm;

   combinatorics::tdo_scheme_compute *TDO;

   flag_orbits_incidence_structure *Flags;
      // if !f_projective_space
   flag_orbits_incidence_structure *Anti_Flags;
```

```
      // if !f_projective_space

   object_with_properties();
   ~object_with_properties();
   void init(
         geometry::object_with_canonical_form *OwCF,
         data_structures::nauty_output *NO,
         int f_projective_space,
         projective_geometry::projective_space_with_action *PA,
         int max_TDO_depth,
         std::string &label,
         int verbose_level);
   void compute_flag_orbits(int verbose_level);
   void lift_generators_to_matrix_group(int verbose_level);
   void init_object_in_projective_space(
         geometry::object_with_canonical_form *OwCF,
         data_structures::nauty_output *NO,
         projective_geometry::projective_space_with_action *PA,
         std::string &label,
         int verbose_level);
   void latex_report(std::ostream &ost,
         combinatorics::classification_of_objects_report_options
            *Report_options,
         int verbose_level);
   void compute_TDO(int max_TDO_depth, int verbose_level);
   void print_TDO(
         std::ostream &ost,
         combinatorics::classification_of_objects_report_options
            *Report_options);
   void export_TDA_with_flag_orbits(
         std::ostream &ost,
         groups::schreier *Sch,
         int verbose_level);
   void export_INP_with_flag_orbits(
         std::ostream &ost,
         groups::schreier *Sch,
         int verbose_level);

};




// ##############################################################################
// regular_linear_space_description.cpp
// ##############################################################################
```

```
//! a description of a class of regular linear spaces from the command line


class regular_linear_space_description {
public:

   int f_m;
   int m;
   int f_n;
   int n;
   int f_k;
   int k;
   int f_r;
   int r;
   int f_target_size;
   int target_size;

   int starter_size;
   int *initial_pair_covering;

   int f_has_control;
   poset_classification::poset_classification_control *Control;



   regular_linear_space_description();
   ~regular_linear_space_description();
   void read_arguments_from_string(
         const char *str, int verbose_level);
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);


};



// ##############################################################################
// regular_ls_classify.cpp
// ##############################################################################


//! classification of regular linear spaces
```

```
class regular_ls_classify {

public:

   regular_linear_space_description *Descr;

   int m2;
   int *v1; // [k]

   poset_classification::poset_with_group_action *Poset;
   poset_classification::poset_classification *gen;
   actions::action *A;
   actions::action *A2;
   induced_actions::action_on_k_subsets *Aonk;
      // only a pointer, do not free

   int *row_sum; // [m]
   int *pairs; // [m2]
   int *open_rows; // [m]
   int *open_row_idx; // [m]
   int *open_pairs; // [m2]
   int *open_pair_idx; // [m2]

   regular_ls_classify();
   ~regular_ls_classify();
   void init_and_run(
         regular_linear_space_description *Descr,
         int verbose_level);
   void init_group(int verbose_level);
   void init_action_on_k_subsets(int onk, int verbose_level);
   void init_generator(
         poset_classification::poset_classification_control *Control,
         groups::strong_generators *Strong_gens,
         int verbose_level);
   void early_test_func(long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   void print(std::ostream &ost, long int *S, int len);
   void lifting_prepare_function_new(
         solvers_package::exact_cover *E, int starter_case,
      long int *candidates, int nb_candidates,
      groups::strong_generators *Strong_gens,
      solvers::diophant *&Dio, long int *&col_labels,
```

```
      int &f_ruled_out,
      int verbose_level);
};





// ##############################################################################
// tactical_decomposition.cpp
// ##############################################################################

//! tactical decomposition of an incidence structure with respect to a given grou
p



class tactical_decomposition {
public:

   int set_size;
   int nb_blocks;
   geometry::incidence_structure *Inc;
   int f_combined_action;
   actions::action *A;
   actions::action *A_on_points;
   actions::action *A_on_lines;
   groups::strong_generators * gens;
   data_structures::partitionstack *Stack;
   groups::schreier *Sch;
   groups::schreier *Sch_points;
   groups::schreier *Sch_lines;

   tactical_decomposition();
   ~tactical_decomposition();
   void init(int nb_rows, int nb_cols,
         geometry::incidence_structure *Inc,
         int f_combined_action,
         actions::action *Aut,
         actions::action *A_on_points,
         actions::action *A_on_lines,
         groups::strong_generators * gens,
         int verbose_level);
   void report(int f_enter_math, std::ostream &ost);
```

```
};




}}}

#endif /* ORBITER_SRC_LIB_TOP_LEVEL_COMBINATORICS_TL_COMBINATORICS_H_ */
```

7.5 Geometry

```
// tl_geometry.h
//
// Anton Betten
//
// moved here from top_level.h: July 28, 2018
// top_level started:  September 23 2010
// based on global.h, which was taken from reader.h: 3/22/09


#ifndef ORBITER_SRC_LIB_TOP_LEVEL_GEOMETRY_TL_GEOMETRY_H_
#define ORBITER_SRC_LIB_TOP_LEVEL_GEOMETRY_TL_GEOMETRY_H_


namespace orbiter {
namespace layer5_applications {
namespace apps_geometry {



// ##############################################################################
// arc_generator_description.cpp
// ##############################################################################

//! description of a classification problem of arcs in a geometry


class arc_generator_description {

public:

   int f_control;
   std::string control_label;
   //int f_poset_classification_control;
   //poset_classification::poset_classification_control *Control;

   int f_d;
   int d;
   // d is the maximum number of points per line


   int f_target_size;
   int target_size;
   // desired size of the arc

   int f_conic_test;
   // if TRUE, ensure that no six points lie on a conic
```

```
   int f_test_nb_Eckardt_points;
   int nb_E;
   //algebraic_geometry::surface_domain *Surf;

   int f_affine;

   int f_no_arc_testing;
   int f_has_forbidden_point_set;
   std::string forbidden_point_set_string;

   int f_override_group;
   std::string override_group_label;

   arc_generator_description();
   ~arc_generator_description();
   int read_arguments(
         int argc, std::string *argv, int verbose_level);
   void print();


};


// ##############################################################################
// arc_generator.cpp
// ##############################################################################

//! classification of arcs in desarguesian projective planes


class arc_generator {

public:


   arc_generator_description *Descr;
   projective_geometry::projective_space_with_action *PA;
   poset_classification::poset_classification_control *Control;

   int nb_points_total;
   int nb_affine_lines;



   int *forbidden_points;
   int nb_forbidden_points;
```

```
int *f_is_forbidden;

groups::strong_generators *SG;

poset_classification::poset_with_group_action *Poset;



int *line_type; // [PA->P->N_lines]


poset_classification::poset_classification *gen;




arc_generator();
~arc_generator();
void main(int verbose_level);
void init(
   arc_generator_description *Descr,
   projective_geometry::projective_space_with_action *PA,
   groups::strong_generators *SG,
   int verbose_level);
void prepare_generator(int verbose_level);
void compute_starter(int verbose_level);

int test_nb_Eckardt_points(
      long int *S, int len, int pt, int nb_E,
      int verbose_level);
int conic_test(
      long int *S, int len, int pt, int verbose_level);
void early_test_func(
      long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
   int verbose_level);
void print(int len, long int *S);
void print_set_in_affine_plane(int len, long int *S);
void point_unrank(int *v, int rk);
int point_rank(int *v);
void compute_line_type(
      long int *set, int len, int verbose_level);
void lifting_prepare_function_new(
      solvers_package::exact_cover *E,
   int starter_case,
```

```
      long int *candidates, int nb_candidates,
      groups::strong_generators *Strong_gens,
      solvers::diophant *&Dio, long int *&col_labels,
      int &f_ruled_out,
      int verbose_level);
      // compute the incidence matrix of tangent lines
      // versus candidate points
      // extended by external lines versus candidate points
   void report(
         isomorph::isomorph &Iso,
         int verbose_level);
   void report_do_the_work(
         std::ostream &ost, isomorph::isomorph &Iso,
         int verbose_level);
   void report_decompositions(
         isomorph::isomorph &Iso, std::ostream &ost, int orbit,
      long int *data, int verbose_level);
   void report_stabilizer(
         isomorph::isomorph &Iso, std::ostream &ost, int orbit,
      int verbose_level);
};




// ##############################################################################
// arc_lifting_simeon.cpp
// ##############################################################################


//! arc lifting according to Simeon Ball and Ray Hill


class arc_lifting_simeon {

public:

   int verbose_level;
   int q;
   int d; // largest number of points per line
   int n; // projective dimension
   int k; // size of the arc
   field_theory::finite_field *F;
   int f_projective;
   int f_general;
   int f_affine;
```

```
   int f_semilinear;
   int f_special;

   actions::action *A;
   ring_theory::longinteger_object go;
   int *Elt;
   int *v;
   groups::schreier *Sch;
   poset_classification::poset_with_group_action *Poset;
   poset_classification::poset_classification *Gen;
   geometry::projective_space *P;

   actions::action *A2; // action on the lines
   actions::action *A3; // action on lines restricted to filtered_lines


   arc_lifting_simeon();
   ~arc_lifting_simeon();
   void init(int q, int d, int n, int k,
         int verbose_level);
   void early_test_func(long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   void do_covering_problem(
         data_structures_groups::set_and_stabilizer *SaS);


};





// ##############################################################################
// choose_points_or_lines.cpp
// ##############################################################################

//! classification of objects in projective planes


class choose_points_or_lines {

public:
   std::string label;
```

```
int t0;

void *data;

actions::action *A;
actions::action *A_lines;
actions::action *A2;
   // = A if f_choose_lines is FALSE
   // = A_lines if f_choose_lines is TRUE

int f_choose_lines;
   // TRUE if we are looking for a set of lines
   // FALSE if we are looking for a set of points
int nb_points_or_lines;
   // the size of the set we are looking for

int print_generators_verbose_level;


int *transporter;
   // maps the canonical rep to the favorite rep
int *transporter_inv;
   // maps the favorite rep to the canonical rep


int (*check_function)(int len,
      long int *S, void *data, int verbose_level);

poset_classification::poset_classification *gen;
poset_classification::poset_classification_control *Control;
poset_classification::poset_with_group_action *Poset;

int nb_orbits;
int current_orbit;

int f_has_favorite;
int f_iso_test_only; // do not change to favorite
long int *favorite;
int favorite_size;

int f_has_orbit_select;
int orbit_select;




long int *representative; // [nb_points_or_lines]
```

```
   ring_theory::longinteger_object *stab_order;
   groups::sims *stab;
   groups::strong_generators *Stab_Strong_gens;


   choose_points_or_lines();
   ~choose_points_or_lines();
   void null();
   void freeself();
   void null_representative();
   void free_representative();
   void init(const char *label, void *data,
         actions::action *A, actions::action *A_lines,
      int f_choose_lines,
      int nb_points_or_lines,
      int (*check_function)(int len,
            long int *S, void *data,
            int verbose_level),
      int t0,
      int verbose_level);
   void compute_orbits_from_sims(
         groups::sims *G, int verbose_level);
   void compute_orbits(
         groups::strong_generators *Strong_gens, int verbose_level);
   void choose_orbit(
         int orbit_no, int &f_hit_favorite, int verbose_level);
   int favorite_orbit_representative(
         int *transporter,
      int *transporter_inv,
      long int *the_favorite_representative,
      int verbose_level);
   void print_rep();
   void print_stab();
   int is_in_rep(int a);

};

// ##############################################################################
// classify_cubic_curves.cpp:
// ##############################################################################


//! classification of cubic curves in PG(2,q)


class classify_cubic_curves {
```

```
public:

   int q;
   field_theory::finite_field *F; // do not free
   actions::action *A; // do not free

   cubic_curve_with_action *CCA; // do not free
   algebraic_geometry::cubic_curve *CC; // do not free

   arc_generator *Arc_gen;

   int nb_orbits_on_sets;
   int nb; // number of orbits for which the rank is 9
   int *Idx; // index set of those orbits for which the rank is 9



   invariant_relations::flag_orbits *Flag_orbits;

   int *Po;

   int nb_orbits_on_curves;

   invariant_relations::classification_step *Curves;



   classify_cubic_curves();
   ~classify_cubic_curves();
   void init(
         projective_geometry::projective_space_with_action *PA,
         cubic_curve_with_action *CCA,
         arc_generator_description *Descr,
         int verbose_level);
   void compute_starter(int verbose_level);
   void test_orbits(int verbose_level);
   void downstep(int verbose_level);
   void upstep(int verbose_level);
   void do_classify(int verbose_level);
   int recognize(int *eqn_in,
         int *Elt, int &iso_type, int verbose_level);
   void family1_recognize(int *Iso_type, int verbose_level);
   void family2_recognize(int *Iso_type, int verbose_level);
   void family3_recognize(int *Iso_type, int verbose_level);
   void familyE_recognize(int *Iso_type, int verbose_level);
   void familyH_recognize(int *Iso_type, int verbose_level);
```

```
   void familyG_recognize(int *Iso_type, int verbose_level);
   void report(std::ostream &ost, int verbose_level);

};




// ##############################################################################
// cubic_curve_action.cpp:
// ##############################################################################

//! domain for cubic curves in projective space with automorphism group



class cubic_curve_with_action {

public:

   int q;
   field_theory::finite_field *F; // do not free

   algebraic_geometry::cubic_curve *CC; // do not free

   actions::action *A; // linear group PGGL(3,q)
   actions::action *A2; // linear group PGGL(3,q) acting on lines

   int *Elt1;

   induced_actions::action_on_homogeneous_polynomials *AonHPD_3_3;



   cubic_curve_with_action();
   ~cubic_curve_with_action();
   void init(
         algebraic_geometry::cubic_curve *CC,
         actions::action *A,
         int verbose_level);

};

// ##############################################################################
// hermitian_spreads_classify.cpp
// ##############################################################################
```

```
//! classification of Hermitian spreads


class hermitian_spreads_classify {
public:
   int n;
   int Q;
   int len; // = n + 1
   field_theory::finite_field *F;
   geometry::hermitian *H;

   long int *Pts;
   int nb_pts;
   int *v;
   int *line_type;
   geometry::projective_space *P;
   groups::strong_generators *sg;
   long int **Intersection_sets;
   int sz;
   long int *secants;
   int nb_secants;
   int *Adj;

   actions::action *A;
   actions::action *A2;
   actions::action *A2r;

   poset_classification::poset_classification_control *Control;
   poset_classification::poset_with_group_action *Poset;
   poset_classification::poset_classification *gen;


   hermitian_spreads_classify();
   ~hermitian_spreads_classify();
   void init(int n, int Q, int verbose_level);
   void read_arguments(int argc, std::string *argv);
   void init2(int verbose_level);
   void compute(int depth, int verbose_level);
   void early_test_func(long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
};
```

```
// ##############################################################################
// linear_set_classify.cpp
// ##############################################################################


//! classification of linear sets



class linear_set_classify {
public:
   int s; // s divides n
   int n; // n = s * m
   int m; // = n / s
   int q;
   int Q; // Q = q^s
   int depth;
   int f_semilinear;
   int schreier_depth;
   int f_use_invariant_subset_if_available;
   int f_debug;
   int f_has_extra_test_func;
   int (*extra_test_func)(void *, int len, long int *S,
      void *extra_test_func_data, int verbose_level);
   void *extra_test_func_data;
   int *Basis; // [depth * vector_space_dimension]
   int *base_cols;

   field_theory::finite_field *Fq;
   field_theory::finite_field *FQ;
   field_theory::subfield_structure *SubS;
   geometry::projective_space *P;


   // the groups we need:

   actions::action *Aq; // GL(n,q)

   actions::action *AQ; // GL(m, Q)
```

```
actions::action *A_PGLQ; // PGL(m,Q)

algebra::vector_space *VS;
poset_classification::poset_classification_control *Control1;
poset_classification::poset_with_group_action *Poset1;
poset_classification::poset_classification *Gen;
int vector_space_dimension; // = n

// the generators:

groups::strong_generators *Strong_gens;
   // generators for GL(m,Q) field reduced into GL(n,q)

geometry::desarguesian_spread *D; // n, m, s

int n1; // = s * m1;
int m1; // = m + 1

geometry::desarguesian_spread *D1; // n1, m1, s

int *spread_embedding; // [D->N]

int f_identify;
int k;
int order;
geometry::spread_domain *SD;
spreads::spread_classify *T;



int secondary_level;
int secondary_orbit_at_level;
int secondary_depth;
long int *secondary_candidates;
int secondary_nb_candidates;
int secondary_schreier_depth;

poset_classification::poset_classification_control *Control_stab;
poset_classification::poset_with_group_action *Poset_stab;
poset_classification::poset_classification *Gen_stab;

poset_classification::poset_classification_control *Control2;
poset_classification::poset_with_group_action *Poset2;
poset_classification::poset_classification *Gen2;
int *is_allowed;

linear_set_classify();
```

```
   ~linear_set_classify();
   void init(
      int s, int n, int q,
      std::string &poly_q, std::string &poly_Q,
      int depth, int f_identify, int verbose_level);
   void do_classify(int verbose_level);
   int test_set(int len, long int *S, int verbose_level);
   void compute_intersection_types_at_level(int level,
      int &nb_nodes, int *&Intersection_dimensions,
      int verbose_level);
   void calculate_intersections(int depth, int verbose_level);
   void read_data_file(int depth, int verbose_level);
   void print_orbits_at_level(int level);
   void classify_secondary(int argc, const char **argv,
      int level, int orbit_at_level,
      groups::strong_generators *strong_gens,
      int verbose_level);
   void init_secondary(int argc, const char **argv,
      long int *candidates, int nb_candidates,
      groups::strong_generators *Strong_gens_previous,
      int verbose_level);
   void do_classify_secondary(int verbose_level);
   int test_set_secondary(
         int len, long int *S, int verbose_level);
   void compute_stabilizer_of_linear_set(
         int argc, const char **argv,
      int level, int orbit_at_level,
      groups::strong_generators *&strong_gens,
      int verbose_level);
   void init_compute_stabilizer(int argc, const char **argv,
      int level, int orbit_at_level,
      long int *candidates, int nb_candidates,
      groups::strong_generators *Strong_gens_previous,
      groups::strong_generators *&strong_gens,
      int verbose_level);
   void do_compute_stabilizer(
         int level, int orbit_at_level,
      long int *candidates, int nb_candidates,
      groups::strong_generators *&strong_gens,
      int verbose_level);
   void construct_semifield(
         int orbit_for_W, int verbose_level);

};
```

```
// ##############################################################################
// ovoid_classify_description.cpp
// ##############################################################################


//! description of a problem of classification of ovoids in orthogonal spaces


class ovoid_classify_description {

public:

   int f_control;
   std::string control_label;
   //poset_classification::poset_classification_control *Control;

   int f_epsilon;
   int epsilon; // the type of the quadric (0, 1 or -1)
   int f_d;
   int d; // algebraic dimension

   ovoid_classify_description();
   ~ovoid_classify_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// ovoid_classify.cpp
// ##############################################################################


//! classification of ovoids in orthogonal spaces


class ovoid_classify {

public:

   ovoid_classify_description *Descr;
   poset_classification::poset_classification_control *Control;
   groups::linear_group *LG;
```

```
int m; // Witt index

poset_classification::poset_with_group_action *Poset;
poset_classification::poset_classification *gen;


actions::action *A;


orthogonal_geometry::orthogonal *O;


int N; // = O->nb_points

int *u, *v, *w, *tmp1; // vectors of length d

int nb_sol; // number of solutions so far


geometry::klein_correspondence *K;
int *color_table;
int nb_colors;

int *Pts; // [N * d]
int *Candidates; // [N * d]


ovoid_classify();
~ovoid_classify();
void init(ovoid_classify_description *Descr,
      groups::linear_group *LG,
      int &verbose_level);
void early_test_func(
      long int *S, int len,
   long int *candidates, int nb_candidates,
   long int *good_candidates, int &nb_good_candidates,
   int verbose_level);
void print(
      std::ostream &ost, long int *S, int len);
void make_graphs(
      orbiter_kernel_system::orbiter_data_file *ODF,
      std::string &prefix,
      int f_split, int split_r, int split_m,
      int f_lexorder_test,
      const char *fname_mask,
      int verbose_level);
void make_one_graph(
```

```
        orbiter_kernel_system::orbiter_data_file *ODF,
        std::string &prefix,
        int orbit_idx,
        int f_lexorder_test,
        graph_theory::colored_graph *&CG,
        int verbose_level);
   void create_graph(
        orbiter_kernel_system::orbiter_data_file *ODF,
      int orbit_idx,
      long int *candidates, int nb_candidates,
      graph_theory::colored_graph *&CG,
      int verbose_level);
   void compute_coloring(
        long int *starter, int starter_size,
        long int *candidates, int nb_points,
        int *point_color, int &nb_colors_used, int verbose_level);

};




// ##############################################################################
// polar.cpp
// ##############################################################################


//! the polar space arising from an orthogonal geometry


class polar {
public:
   int epsilon;
   int n; // vector space dimension
   int k;
   int q;
   int depth;

   int f_print_generators;

   actions::action *A; // the orthogonal action



   groups::matrix_group *Mtx;
      // only a copy of a pointer, not to be freed
   orthogonal_geometry::orthogonal *O;
```

```
   // only a copy of a pointer, not to be freed
field_theory::finite_field *F;
   // only a copy of a pointer, not to be freed

int *tmp_M; // [n * n]
int *base_cols; // [n]

algebra::vector_space *VS;
poset_classification::poset_classification_control *Control;
poset_classification::poset_with_group_action *Poset;
poset_classification::poset_classification *Gen;

int schreier_depth;
int f_use_invariant_subset_if_available;
int f_debug;

int f_has_strong_generators;
int f_has_strong_generators_allocated;
groups::strong_generators *Strong_gens;

int first_node, nb_orbits, nb_elements;

polar();
~polar();
void init_group_by_base_images(
      int *group_generator_data,
   int group_generator_size,
   int f_group_order_target, const char *group_order_target,
   int verbose_level);
void init_group(
      int *group_generator_data, int group_generator_size,
   int f_group_order_target, const char *group_order_target,
   int verbose_level);
void init(
      actions::action *A,
      orthogonal_geometry::orthogonal *O,
   int epsilon, int n, int k,
   field_theory::finite_field *F, int depth,
   int verbose_level);
void init2(int depth, int verbose_level);
void compute_orbits(
      int t0, int verbose_level);
void compute_cosets(
      int depth, int orbit_idx, int verbose_level);
void dual_polar_graph(
      int depth, int orbit_idx,
      ring_theory::longinteger_object *&Rank_table,
```

```
         int &nb_maximals,
      int verbose_level);
   void show_stabilizer(
         int depth, int orbit_idx, int verbose_level);
   void test_if_in_perp(
         long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   void test_if_closed_under_cosets(
         int *S, int len,
      int *candidates, int nb_candidates,
      int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   void get_stabilizer(
         int orbit_idx,
         data_structures_groups::group_container &G,
         ring_theory::longinteger_object &go_G);
   void get_orbit_length(
         int orbit_idx, ring_theory::longinteger_object &length);
   int get_orbit_length_as_int(int orbit_idx);
   void orbit_element_unrank(int orbit_idx, long int rank,
      long int *set, int verbose_level);
   void orbit_element_rank(int &orbit_idx, long int &rank,
      long int *set, int verbose_level);
   void unrank_point(int *v, int rk);
   int rank_point(int *v);
   void list_whole_orbit(
         int depth, int orbit_idx, int f_limit, int limit);
};




// ##############################################################################
// search_blocking_set.cpp
// ##############################################################################

//! classification of blocking sets in projective planes



class search_blocking_set {
public:
   geometry::incidence_structure *Inc; // do not free
   actions::action *A; // do not free
   poset_classification::poset_classification_control *Control;
```

```
   poset_classification::poset_with_group_action *Poset;
   poset_classification::poset_classification *gen;

   data_structures::fancy_set *Line_intersections; // [Inc->nb_cols]
   long int *blocking_set;
   int blocking_set_len;
   int *sz; // [Inc->nb_cols]

   data_structures::fancy_set *active_set;
   int *sz_active_set; // [Inc->nb_cols + 1]

   std::deque<std::vector<int> > solutions;
   int nb_solutions;
   int f_find_only_one;
   int f_blocking_set_size_desired;
   int blocking_set_size_desired;

   int max_search_depth;
   int *search_nb_candidates;
   int *search_cur;
   int **search_candidates;
   int **save_sz;


   search_blocking_set();
   ~search_blocking_set();
   void init(
         geometry::incidence_structure *Inc,
         actions::action *A,
         int verbose_level);
   void find_partial_blocking_sets(
         int depth, int verbose_level);
   int test_level(int depth, int verbose_level);
   int test_blocking_set(
         int len, long int *S, int verbose_level);
   int test_blocking_set_upper_bound_only(
         int len, long int *S,
      int verbose_level);
   void search_for_blocking_set(int input_no,
      int level, int f_all, int verbose_level);
   int recursive_search_for_blocking_set(int input_no,
      int starter_level, int level, int verbose_level);
   void save_line_intersection_size(int level);
   void restore_line_intersection_size(int level);
};
```

```
// ##############################################################################
// singer_cycle.cpp
// ##############################################################################

//! the Singer cycle in a finite projective geometry


class singer_cycle {
public:
   field_theory::finite_field *F;
   actions::action *A;
   actions::action *A2;
   int n;
   int q;
   int *poly_coeffs; // of degree n
   int *Singer_matrix;
   data_structures_groups::vector_ge *nice_gens;
   groups::strong_generators *SG;
   ring_theory::longinteger_object target_go;
   geometry::projective_space *P;
   int *singer_point_list;
   int *singer_point_list_inv;
   groups::schreier *Sch;
   int nb_line_orbits;
   int *line_orbit_reps;
   int *line_orbit_len;
   int *line_orbit_first;
   std::string *line_orbit_label;
   std::string *line_orbit_label_tex;
   int *line_orbit;
   int *line_orbit_inv;
   geometry::incidence_structure *Inc;
   apps_combinatorics::tactical_decomposition *T;

   singer_cycle();
   ~singer_cycle();
   void init(int n, field_theory::finite_field *F, actions::action *A,
         actions::action *A2, int verbose_level);
   void init_lines(int verbose_level);
};




// ##############################################################################
// tensor_classify.cpp
// ##############################################################################
```

```
//! classification of tensors under the wreath product group


class tensor_classify {
public:
   int t0;

   int nb_factors;
   int n;
   int q;

   field_theory::finite_field *F;
   actions::action *A;
   actions::action *A0;

   actions::action *Ar;
   int nb_points;
   long int *points;


   groups::strong_generators *SG;
   ring_theory::longinteger_object go;
   groups::wreath_product *W;
   algebra::vector_space *VS;
   poset_classification::poset_classification_control *Control;
   poset_classification::poset_with_group_action *Poset;
   poset_classification::poset_classification *Gen;
   int vector_space_dimension;
   int *v; // [vector_space_dimension]

   tensor_classify();
   ~tensor_classify();
   void init(
         field_theory::finite_field *F, groups::linear_group *LG,
         int verbose_level);
   void classify_poset(int depth,
         poset_classification::poset_classification_control *Control,
         int verbose_level);
   void create_restricted_action_on_rank_one_tensors(
         int verbose_level);
   void early_test_func(long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   void report(int f_poset_classify, int poset_classify_depth,
         graphics::layered_graph_draw_options *draw_options,
```

```
        int verbose_level);
};









// ##############################################################################
// top_level_geometry_global.cpp
// ##############################################################################



//! catch all class for geometry




class top_level_geometry_global {
public:

   top_level_geometry_global();
   ~top_level_geometry_global();
   void set_stabilizer_projective_space(
         projective_geometry::projective_space_with_action *PA,
         int intermediate_subset_size,
         std::string &fname_mask, int nb, std::string &column_label,
         std::string &fname_out,
         int verbose_level);
   void report_decomposition_by_group(
         projective_geometry::projective_space_with_action *PA,
         groups::strong_generators *SG, std::ostream &ost, std::string &fname_base,
         int verbose_level);
   void report_decomposition_by_single_automorphism(
         projective_geometry::projective_space_with_action *PA,
         int *Elt, std::ostream &ost, std::string &fname_base,
         int verbose_level);
};



}}}
```

```
#endif /* ORBITER_SRC_LIB_TOP_LEVEL_GEOMETRY_TL_GEOMETRY_H_ */
```

7.6 Graph Theory

```
/*
 * graph_theory.h
 *
 *  Created on: Mar 30, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_GRAPH_THEORY_GRAPH_THEORY_H_
#define SRC_LIB_TOP_LEVEL_GRAPH_THEORY_GRAPH_THEORY_H_




namespace orbiter {
namespace layer5_applications {
namespace apps_graph_theory {


// ##############################################################################
// cayley_graph_search.cpp
// ##############################################################################

//! for a problem of Ferdinand Ihringer


class cayley_graph_search {

public:

   int level;
   int group;
   int subgroup;

   int ord;
   int degree;
   int data_size;

   long int go;
   int go_subgroup;
   int nb_involutions;
   int *f_has_order2;
   int *f_subgroup; // [go]
   int *list_of_elements; // [go]
   int *list_of_elements_inverse; // [go]

   actions::action *A;
```

```
field_theory::finite_field *F;
int target_depth;

int *Elt1;
int *Elt2;
data_structures_groups::vector_ge *gens;
data_structures_groups::vector_ge *gens_subgroup;
ring_theory::longinteger_object target_go, target_go_subgroup;
groups::strong_generators *Strong_gens;
groups::strong_generators *Strong_gens_subgroup;

groups::sims *S;
groups::sims *S_subgroup;

int *Table;
int *generators;
int nb_generators;

std::string fname_base;
std::string prefix;
std::string fname;
std::string fname_graphs;

groups::strong_generators *Aut_gens;
ring_theory::longinteger_object Aut_order;
actions::action *Aut;
actions::action *A2;
poset_classification::poset_with_group_action *Poset;
poset_classification::poset_classification_control *Control;
poset_classification::poset_classification *gen;


void init(int level, int group, int subgroup, int verbose_level);
void init_group(int verbose_level);
void init_group2(int verbose_level);
void init_group_level_3(int verbose_level);
void init_group_level_4(int verbose_level);
void init_group_level_5(int verbose_level);
int incremental_check_func(int len, long int *S, int verbose_level);
void classify_subsets(int verbose_level);
void write_file(int verbose_level);
void create_Adjacency_list(long int *Adj,
   long int *connection_set, int connection_set_sz,
   int verbose_level);
// Adj[go * connection_set_sz]
void create_additional_edges(
   long int *Additional_neighbor,
```

```
      int *Additional_neighbor_sz,
      long int connection_element,
      int verbose_level);
   // Additional_neighbor[go], Additional_neighbor_sz[go]

};


// ##############################################################################
// create_graph_description.cpp
// ##############################################################################


//! a description of a graph using command line arguments


class create_graph_description {
public:

   int f_load;
   std::string fname;

   int f_load_csv_no_border;

   int f_load_adjacency_matrix_from_csv_and_select_value;
   std::string load_adjacency_matrix_from_csv_and_select_value_fname;
   int load_adjacency_matrix_from_csv_and_select_value_value;

   int f_load_dimacs;

   int f_edge_list;
   int n;
   std::string edge_list_text;

   int f_edges_as_pairs;
   std::string edges_as_pairs_text;

   int f_cycle;
   int cycle_n;

   int f_inversion_graph;
   std::string inversion_graph_text;

   int f_Hamming;
   int Hamming_n;
   int Hamming_q;
```

```
int f_Johnson;
int Johnson_n;
int Johnson_k;
int Johnson_s;

int f_Paley;
std::string Paley_label_Fq;

int f_Sarnak;
int Sarnak_p;
int Sarnak_q;

int f_Schlaefli;
std::string Schlaefli_label_Fq;

int f_Shrikhande;

int f_Winnie_Li;
std::string Winnie_Li_label_Fq;
int Winnie_Li_index;

int f_Grassmann;
int Grassmann_n;
int Grassmann_k;
std::string Grassmann_label_Fq;
int Grassmann_r;

int f_coll_orthogonal;
int coll_orthogonal_epsilon;
int coll_orthogonal_d;
std::string coll_orthogonal_label_Fq;

int f_trihedral_pair_disjointness_graph;

int f_non_attacking_queens_graph;
int non_attacking_queens_graph_n;

int f_subset;
std::string subset_label;
std::string subset_label_tex;
std::string subset_text;

int f_disjoint_sets_graph;
std::string disjoint_sets_graph_fname;

int f_orbital_graph;
std::string orbital_graph_group;
```

```
   int orbital_graph_orbit_idx;

   int f_collinearity_graph;
   std::string collinearity_graph_matrix;

   int f_chain_graph;
   std::string chain_graph_partition_1;
   std::string chain_graph_partition_2;

   int f_Cayley_graph;
   std::string Cayley_graph_group;
   std::string Cayley_graph_gens;

   std::vector<graph_modification_description> Modifications;

   create_graph_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();


};


// ##############################################################################
// create_graph.cpp
// ##############################################################################


//! creates a graph from a description with create_graph_description


class create_graph {
public:

   create_graph_description *description;

   int f_has_CG;
   graph_theory::colored_graph *CG;

   int N;
   int *Adj;


   std::string label;
   std::string label_tex;
```

```
   create_graph();
   ~create_graph();
   void init(
         create_graph_description *description,
         int verbose_level);
   void create_cycle(
         int n, int verbose_level);
   void create_inversion_graph(
         std::string &perm_text, int verbose_level);
   void create_Hamming(
         int n, int q, int verbose_level);
   void create_Johnson(
         int n, int k, int s, int verbose_level);
   void create_Paley(
         std::string &label_Fq, int verbose_level);
   void create_Sarnak(
         int p, int q,
         int verbose_level);
   void create_Schlaefli(
         std::string &label_Fq, int verbose_level);
   void create_Shrikhande(
         int verbose_level);
   void create_Winnie_Li(
         std::string &label_Fq, int index, int verbose_level);
   void create_Grassmann(
         int n, int k, std::string &label_Fq,
         int r, int verbose_level);
   void create_coll_orthogonal(
         int epsilon, int d, std::string &label_Fq,
         int verbose_level);
   void make_orbital_graph(
         apps_algebra::any_group *AG, int orbit_idx,
         int verbose_level);
   void make_collinearity_graph(
         int *Inc, int nb_rows, int nb_cols,
         int verbose_level);
   void make_chain_graph(
         int *part1, int sz1,
         int *part2, int sz2,
         int verbose_level);

};


// ##############################################################################
// graph_classification_activity_description.cpp
```

```
// ##############################################################################

//! an activity for a classification of graphs and tournaments


class graph_classification_activity_description {

public:
   int f_draw_level_graph;
   int draw_level_graph_level;

   int f_draw_graphs;

   int f_list_graphs_at_level;
   int list_graphs_at_level_level_min;
   int list_graphs_at_level_level_max;

   int f_draw_graphs_at_level;
   int draw_graphs_at_level_level;

   int f_draw_options;
   graphics::layered_graph_draw_options *draw_options;

   int f_recognize_graphs_from_adjacency_matrix_csv;
   std::string recognize_graphs_from_adjacency_matrix_csv_fname;


   graph_classification_activity_description();
   ~graph_classification_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// graph_classification_activity.cpp
// ##############################################################################

//! an activity for a classification of graphs and tournaments


class graph_classification_activity {

public:
```

```
   graph_classification_activity_description *Descr;
   graph_classify *GC;

   graph_classification_activity();
   ~graph_classification_activity();
   void init(graph_classification_activity_description *Descr,
         graph_classify *GC,
         int verbose_level);
   void perform_activity(int verbose_level);


};


// ##############################################################################
// graph_classify_description.cpp
// ##############################################################################

//! classification of graphs and tournaments


class graph_classify_description {

public:
   int f_n;
   int n; // number of vertices

   int f_regular;

   int f_control;
   poset_classification::poset_classification_control *Control;

   int regularity;

   int f_girth;
   int girth;

   int f_depth;
   int depth;

   int f_tournament;
   int f_no_superking;

   int f_test_multi_edge;
```

```
   int f_identify;
   long int identify_data[1000];
   int identify_data_sz;


   graph_classify_description();
   ~graph_classify_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// graph_classify.cpp
// ##############################################################################

//! classification of graphs and tournaments


class graph_classify {

public:

   graph_classify_description *Descr;

   poset_classification::poset_with_group_action *Poset;
   poset_classification::poset_classification *gen;

   actions::action *A_base; // symmetric group on n vertices
   actions::action *A_on_edges; // action on pairs

   int n2; // n choose 2

   int *adjacency; // [n * n]

   int *degree_sequence; // [n]

   int *neighbor; // [n]
   int *neighbor_idx; // [n]
   int *distance; // [n]

   long int *S1; // [n2]
```

```
   graph_classify();
   ~graph_classify();
   void init(
         graph_classify_description *Descr, int verbose_level);
   int check_conditions(int len, long int *S, int verbose_level);
   int check_conditions_tournament(int len, long int *S,
         int verbose_level);
   int check_regularity(long int *S, int len,
         int verbose_level);
   int compute_degree_sequence(
         long int *S, int len);
   int girth_check(long int *S, int len, int verbose_level);
   int girth_test_vertex(long int *S, int len,
         int vertex, int girth, int verbose_level);
   void get_adjacency(long int *S, int len, int verbose_level);
   void print(std::ostream &ost, long int *S, int len);
   void print_score_sequences(int level, int verbose_level);
   void score_sequence(
         int n, long int *set, int sz, long int *score,
         int verbose_level);
   void list_graphs(int level_min, int level_max, int verbose_level);
   void draw_graphs(int level,
         graphics::layered_graph_draw_options *draw_options,
         int verbose_level);
   void recognize_graph_from_adjacency_list(int *Adj, int N2,
         int &iso_type,
         int verbose_level);
   int number_of_orbits();

};



// ##############################################################################
// graph_modification_description.cpp
// ##############################################################################

//! unary operators to modify graphs and tournaments


class graph_modification_description {

public:
```

```
   int f_complement;

   int f_distance_2;


   graph_modification_description();
   ~graph_modification_description();
   int check_and_parse_argument(
      int argc, int &i, std::string *argv,
      int verbose_level);
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();
   void apply(graph_theory::colored_graph *&CG, int verbose_level);

};


// ##############################################################################
// graph_theoretic_activity_description.cpp
// ##############################################################################

//! description of an activity for graphs


class graph_theoretic_activity_description {

public:

   int f_find_cliques;
   graph_theory::clique_finder_control *Clique_finder_control;

   int f_find_subgraph;
   std::string find_subgraph_label;

   int f_export_magma;
   int f_export_maple;
   int f_export_csv;
   int f_export_graphviz;
   int f_print;
   int f_sort_by_colors;

   int f_split;
   std::string split_input_fname;
   std::string split_by_file;
```

```
   int f_split_by_starters;
   std::string split_by_starters_fname_reps;
   std::string split_by_starters_col_label;

   int f_split_by_clique;
   std::string split_by_clique_label;
   std::string split_by_clique_set;

   int f_save;
   int f_automorphism_group;

   int f_properties;
   int f_eigenvalues;
   int f_draw;


   graph_theoretic_activity_description();
   ~graph_theoretic_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();


};


// ##############################################################################
// graph_theoretic_activity.cpp
// ##############################################################################

//! an activity for graphs


class graph_theoretic_activity {

public:

   graph_theoretic_activity_description *Descr;
   graph_theory::colored_graph *CG;


   graph_theoretic_activity();
   ~graph_theoretic_activity();
   void init(graph_theoretic_activity_description *Descr,
         graph_theory::colored_graph *CG,
         int verbose_level);
```

```
   void perform_activity(int verbose_level);


};




// ##############################################################################
// graph_theory_apps.cpp
// ##############################################################################

//! applications in graph theory involving groups


class graph_theory_apps {

public:

   graph_theory_apps();
   ~graph_theory_apps();
   void automorphism_group(
         graph_theory::colored_graph *CG, int verbose_level);
   void expander_graph(
         int p, int q,
         int f_special,
         field_theory::finite_field *F,
         actions::action *A,
         int *&Adj, int &N,
         int verbose_level);


};




}}}


#endif /* SRC_LIB_TOP_LEVEL_GRAPH_THEORY_GRAPH_THEORY_H_ */
```

7.7 User Interface

```
/*
 * interfaces.h
 *
 *  Created on: Apr 3, 2020
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_INTERFACES_INTERFACES_H_
#define SRC_LIB_TOP_LEVEL_INTERFACES_INTERFACES_H_




namespace orbiter {
namespace layer5_applications {
namespace user_interface {


// ##############################################################################
// activity_description.cpp
// ##############################################################################

//! description of an activity for an orbiter symbol


class activity_description {

   interface_symbol_table *Sym;


   int f_finite_field_activity;
   field_theory::finite_field_activity_description
      *Finite_field_activity_description;

   int f_polynomial_ring_activity;
   ring_theory::polynomial_ring_activity_description
      *Polynomial_ring_activity_description;

   int f_projective_space_activity;
   projective_geometry::projective_space_activity_description
      *Projective_space_activity_description;

   int f_orthogonal_space_activity;
   orthogonal_geometry_applications::orthogonal_space_activity_description
      *Orthogonal_space_activity_description;
```

```
int f_group_theoretic_activity;
apps_algebra::group_theoretic_activity_description
   *Group_theoretic_activity_description;

int f_coding_theoretic_activity;
apps_coding_theory::coding_theoretic_activity_description
   *Coding_theoretic_activity_description;

int f_cubic_surface_activity;
applications_in_algebraic_geometry::cubic_surfaces_in_general::cubic_surface_ac
tivity_description
   *Cubic_surface_activity_description;

int f_quartic_curve_activity;
applications_in_algebraic_geometry::quartic_curves::quartic_curve_activity_desc
ription
   *Quartic_curve_activity_description;

int f_blt_set_activity;
orthogonal_geometry_applications::blt_set_activity_description
   *Blt_set_activity_description;

int f_combinatorial_object_activity;
apps_combinatorics::combinatorial_object_activity_description
   *Combinatorial_object_activity_description;

int f_graph_theoretic_activity;
apps_graph_theory::graph_theoretic_activity_description
   *Graph_theoretic_activity_description;

int f_classification_of_cubic_surfaces_with_double_sixes_activity;
applications_in_algebraic_geometry::cubic_surfaces_and_double_sixes::classifica
tion_of_cubic_surfaces_with_double_sixes_activity_description
   *Classification_of_cubic_surfaces_with_double_sixes_activity_description;

int f_spread_table_activity;
spreads::spread_table_activity_description
   *Spread_table_activity_description;

int f_packing_with_symmetry_assumption_activity;
packings::packing_was_activity_description
   *Packing_was_activity_description;

int f_packing_fixed_points_activity;
packings::packing_was_fixpoints_activity_description
   *Packing_was_fixpoints_activity_description;
```

```
   int f_graph_classification_activity;
   apps_graph_theory::graph_classification_activity_description
      *Graph_classification_activity_description;

   int f_diophant_activity;
   solvers::diophant_activity_description
      *Diophant_activity_description;

   int f_design_activity;
   apps_combinatorics::design_activity_description
      *Design_activity_description;

   int f_large_set_was_activity;
   apps_combinatorics::large_set_was_activity_description
      *Large_set_was_activity_description;

   int f_formula_activity;
   expression_parser::formula_activity_description
      *Formula_activity_description;

   int f_BLT_set_classify_activity;
   orthogonal_geometry_applications::blt_set_classify_activity_description
      *Blt_set_classify_activity_description;

   int f_spread_classify_activity;
   spreads::spread_classify_activity_description
      *Spread_classify_activity_description;

   int f_spread_activity;
   spreads::spread_activity_description
      *Spread_activity_description;

   int f_translation_plane_activity;
   spreads::translation_plane_activity_description
      *Translation_plane_activity_description;

   int f_action_on_forms_activity;
   apps_algebra::action_on_forms_activity_description
      *Action_on_forms_activity_description;

   int f_orbits_activity;
   apps_algebra::orbits_activity_description
      *Orbits_activity_description;

public:
   activity_description();
   ~activity_description();
```

```
   void read_arguments(
         interface_symbol_table *Sym,
         int argc, std::string *argv, int &i, int verbose_level);
   void worker(int verbose_level);
   void print();
   void do_finite_field_activity(int verbose_level);
   void do_ring_theoretic_activity(int verbose_level);
   void do_projective_space_activity(int verbose_level);
   void do_orthogonal_space_activity(int verbose_level);
   void do_group_theoretic_activity(int verbose_level);
   void do_coding_theoretic_activity(int verbose_level);
   void do_cubic_surface_activity(int verbose_level);
   void do_quartic_curve_activity(int verbose_level);
   void do_blt_set_activity(int verbose_level);
   void do_combinatorial_object_activity(int verbose_level);
   void do_graph_theoretic_activity(int verbose_level);
   void do_classification_of_cubic_surfaces_with_double_sixes_activity(
         int verbose_level);
   void do_spread_table_activity(int verbose_level);
   void do_packing_was_activity(int verbose_level);
   void do_packing_fixed_points_activity(int verbose_level);
   void do_graph_classification_activity(int verbose_level);
   void do_diophant_activity(int verbose_level);
   void do_design_activity(int verbose_level);
   void do_large_set_was_activity(int verbose_level);
   void do_formula_activity(int verbose_level);
   void do_BLT_set_classify_activity(int verbose_level);
   void do_spread_classify_activity(int verbose_level);
   void do_spread_activity(int verbose_level);
   void do_translation_plane_activity(int verbose_level);
   void do_action_on_forms_activity(int verbose_level);
   void do_orbits_activity(int verbose_level);

};



// ##############################################################################
// interface_algebra.cpp
// ##############################################################################

//! interface to the algebra module


class interface_algebra {

   int f_count_subprimitive;
```

```
int count_subprimitive_Q_max;
int count_subprimitive_H_max;

int f_character_table_symmetric_group;
int character_table_symmetric_group_n;

int f_make_A5_in_PSL_2_q;
int make_A5_in_PSL_2_q_q;

int f_order_of_q_mod_n;
int order_of_q_mod_n_q;
int order_of_q_mod_n_n_min;
int order_of_q_mod_n_n_max;

int f_eulerfunction_interval;
int eulerfunction_interval_n_min;
int eulerfunction_interval_n_max;

int f_young_symmetrizer;
int young_symmetrizer_n;
int f_young_symmetrizer_sym_4;

int f_draw_mod_n;
graphics::draw_mod_n_description *Draw_mod_n_description;

int f_power_function_mod_n;
int power_function_mod_n_k;
int power_function_mod_n_n;


// the following two cannot be finite field activities because
// finite field activities are at layer 1 and these functions require level 5.

// perhaps they should be projective space activities,
// because they need a general linear group

int f_all_rational_normal_forms;
std::string all_rational_normal_forms_finite_field_label;
int all_rational_normal_forms_d;

int f_eigenstuff;
std::string eigenstuff_finite_field_label;
int eigenstuff_n;
std::string eigenstuff_coeffs;
std::string eigenstuff_fname;
```

```
public:
   interface_algebra();
   void print_help(int argc,
         std::string *argv, int i, int verbose_level);
   int recognize_keyword(int argc,
         std::string *argv, int i, int verbose_level);
   void read_arguments(int argc,
         std::string *argv, int &i, int verbose_level);
   void print();
   void worker(int verbose_level);
   void do_character_table_symmetric_group(int deg, int verbose_level);

};



// ##############################################################################
// interface_coding_theory.cpp
// ##############################################################################

//! interface to the coding theory module


class interface_coding_theory {

   int f_make_macwilliams_system;
   int make_macwilliams_system_q;
   int make_macwilliams_system_n;
   int make_macwilliams_system_k;

   int f_table_of_bounds;
   int table_of_bounds_n_max;
   int table_of_bounds_q;

   int f_make_bounds_for_d_given_n_and_k_and_q;
   int make_bounds_n;
   int make_bounds_k;
   int make_bounds_q;

   int f_introduce_errors;
   coding_theory::crc_options_description
      *introduce_errors_crc_options_description;

   int f_check_errors;
   coding_theory::crc_options_description
      *check_errors_crc_options_description;
```

```
   int f_extract_block;
   coding_theory::crc_options_description
      *extract_block_crc_options_description;

   int f_random_noise_in_bitmap_file;
   std::string random_noise_in_bitmap_file_input;
   std::string random_noise_in_bitmap_file_output;
   int random_noise_in_bitmap_file_numerator;
   int random_noise_in_bitmap_file_denominator;

   int f_random_noise_of_burst_type_in_bitmap_file;
   std::string random_noise_of_burst_type_in_bitmap_file_input;
   std::string random_noise_of_burst_type_in_bitmap_file_output;
   int random_noise_of_burst_type_in_bitmap_file_numerator;
   int random_noise_of_burst_type_in_bitmap_file_denominator;
   int random_noise_of_burst_type_in_bitmap_file_burst_length;

public:
   interface_coding_theory();
   void print_help(int argc, std::string *argv, int i, int verbose_level);
   int recognize_keyword(int argc, std::string *argv, int i, int verbose_level);
   void read_arguments(int argc,
         std::string *argv, int &i, int verbose_level);
   void print();
   void worker(int verbose_level);
};


// ##############################################################################
// interface_combinatorics.cpp
// ##############################################################################

//! interface to the coding theory module


class interface_combinatorics {

   int f_diophant;
   solvers::diophant_description *Diophant_description;

   int f_diophant_activity;
   solvers::diophant_activity_description *Diophant_activity_description;

   int f_random_permutation;
   int random_permutation_degree;
   std::string random_permutation_fname_csv;
```

```
int f_create_random_k_subsets;
int create_random_k_subsets_n;
int create_random_k_subsets_k;
int create_random_k_subsets_nb;

int f_read_poset_file;
std::string read_poset_file_fname;

int f_grouping;
double grouping_x_stretch;

int f_list_parameters_of_SRG;
int list_parameters_of_SRG_v_max;

int f_conjugacy_classes_Sym_n;
int conjugacy_classes_Sym_n_n;

int f_tree_of_all_k_subsets;
int tree_of_all_k_subsets_n;
int tree_of_all_k_subsets_k;

int f_Delandtsheer_Doyen;
apps_combinatorics::delandtsheer_doyen_description
   *Delandtsheer_Doyen_description;

int f_tdo_refinement;
combinatorics::tdo_refinement_description
   *Tdo_refinement_descr;

int f_tdo_print;
std::string tdo_print_fname;

int f_convert_stack_to_tdo;
std::string stack_fname;

int f_maximal_arc_parameters;
int maximal_arc_parameters_q, maximal_arc_parameters_r;

int f_arc_parameters;
int arc_parameters_q, arc_parameters_s, arc_parameters_r;

int f_pentomino_puzzle;

int f_regular_linear_space_classify;
apps_combinatorics::regular_linear_space_description *Rls_descr;

int f_draw_layered_graph;
```

```
   std::string draw_layered_graph_fname;
   graphics::layered_graph_draw_options *Layered_graph_draw_options;

   int f_domino_portrait;
   int domino_portrait_D;
   int domino_portrait_s;
   std::string domino_portrait_fname;
   graphics::layered_graph_draw_options *domino_portrait_draw_options;

   int f_read_solutions_and_tally;
   std::string read_solutions_and_tally_fname;
   int read_solutions_and_tally_sz;

   int f_make_elementary_symmetric_functions;
   int make_elementary_symmetric_functions_n;
   int make_elementary_symmetric_functions_k_max;

   int f_Dedekind_numbers;
   int Dedekind_n_min;
   int Dedekind_n_max;
   int Dedekind_q_min;
   int Dedekind_q_max;

   int f_rank_k_subset;
   int rank_k_subset_n;
   int rank_k_subset_k;
   std::string rank_k_subset_text;

   int f_geometry_builder;
   geometry_builder::geometry_builder_description
      *Geometry_builder_description;

   int f_union;
   std::string union_set_of_sets_fname;
   std::string union_input_fname;
   std::string union_output_fname;


public:
   interface_combinatorics();
   void print_help(int argc,
         std::string *argv, int i, int verbose_level);
   int recognize_keyword(int argc,
         std::string *argv, int i, int verbose_level);
   void read_arguments(int argc,
         std::string *argv, int &i, int verbose_level);
   void print();
```

```
   void worker(int verbose_level);
   void do_diophant(
         solvers::diophant_description *Descr,
         int verbose_level);
   void do_diophant_activity(
         solvers::diophant_activity_description *Descr,
         int verbose_level);
   void do_conjugacy_classes_Sym_n(int n, int verbose_level);
   void do_conjugacy_classes_Sym_n_file(int n, int verbose_level);
   void do_Delandtsheer_Doyen(
         apps_combinatorics::delandtsheer_doyen_description *Descr,
         int verbose_level);

};


// ##############################################################################
// interface_cryptography.cpp
// ##############################################################################


enum cipher_type { no_cipher_type, substitution, vigenere, affine };

typedef enum cipher_type cipher_type;

//! interface to the cryptography module

class interface_cryptography {

   int f_cipher;
   cipher_type t;
   int f_decipher;
   int f_analyze;
   int f_kasiski;
   int f_avk;
   int key_length, threshold;
   int affine_a;
   int affine_b;

   std::string ptext;
   std::string ctext;
   std::string guess;
   std::string key;

   int f_RSA;
   long int RSA_d;
   long int RSA_m;
```

```
std::string RSA_text;

int f_primitive_root;
std::string primitive_root_p;

int f_smallest_primitive_root;
int smallest_primitive_root_p;

int f_smallest_primitive_root_interval;
int smallest_primitive_root_interval_min;
int smallest_primitive_root_interval_max;

int f_number_of_primitive_roots_interval;

int f_inverse_mod;
int inverse_mod_a;
int inverse_mod_n;

int f_extended_gcd;
int extended_gcd_a;
int extended_gcd_b;

int f_power_mod;
std::string power_mod_a;
std::string power_mod_k;
std::string power_mod_n;

int f_discrete_log;
long int discrete_log_y;
long int discrete_log_a;
long int discrete_log_m;

int f_RSA_setup;
int RSA_setup_nb_bits;
int RSA_setup_nb_tests_solovay_strassen;
int RSA_setup_f_miller_rabin_test;

int f_RSA_encrypt_text;
int RSA_block_size;
std::string RSA_encrypt_text;

int f_sift_smooth;
int sift_smooth_from;
int sift_smooth_len;
std::string sift_smooth_factor_base;

int f_square_root;
```

```
std::string square_root_number;

int f_square_root_mod;
std::string square_root_mod_a;
std::string square_root_mod_m;

int f_all_square_roots_mod_n;
std::string all_square_roots_mod_n_a;
std::string all_square_roots_mod_n_n;

int f_quadratic_sieve;
int quadratic_sieve_n;
int quadratic_sieve_factorbase;
int quadratic_sieve_x0;

int f_jacobi;
long int jacobi_top;
long int jacobi_bottom;

int f_solovay_strassen;
int solovay_strassen_p;
int solovay_strassen_a;

int f_miller_rabin;
int miller_rabin_p;
int miller_rabin_nb_times;

int f_fermat_test;
int fermat_test_p;
int fermat_test_nb_times;

int f_find_pseudoprime;
int find_pseudoprime_nb_digits;
int find_pseudoprime_nb_fermat;
int find_pseudoprime_nb_miller_rabin;
int find_pseudoprime_nb_solovay_strassen;

int f_find_strong_pseudoprime;

int f_miller_rabin_text;
int miller_rabin_text_nb_times;
std::string miller_rabin_number_text;

int f_random;
int random_nb;
std::string random_fname_csv;
```

```
   int f_random_last;
   int random_last_nb;

   int f_affine_sequence;
   int affine_sequence_a;
   int affine_sequence_c;
   int affine_sequence_m;

   int f_Chinese_remainders;
   std::string Chinese_remainders_R;
   std::string Chinese_remainders_M;


public:
   interface_cryptography();
   void print_help(int argc,
         std::string *argv, int i, int verbose_level);
   int recognize_keyword(int argc,
         std::string *argv, int i, int verbose_level);
   void read_arguments(int argc,
         std::string *argv, int &i, int verbose_level);
   void print();
   void worker(int verbose_level);


};

// ##############################################################################
// interface_povray.cpp
// ##############################################################################

//! interface to the povray rendering module


class interface_povray {

   int f_povray;
   graphics::povray_job_description *Povray_job_description;

   int f_prepare_frames;
   orbiter_kernel_system::prepare_frames *Prepare_frames;

public:
   interface_povray();
   void print_help(int argc,
         std::string *argv, int i, int verbose_level);
   int recognize_keyword(int argc,
```

```
        std::string *argv, int i, int verbose_level);
   void read_arguments(int argc,
        std::string *argv, int &i, int verbose_level);
   void print();
   void worker(int verbose_level);
};




// ##############################################################################
// interface_projective.cpp
// ##############################################################################

//! interface to the projective geometry module


class interface_projective {


   int f_create_points_on_quartic;
   double desired_distance;

   int f_create_points_on_parabola;
   int parabola_N;
   double parabola_a;
   double parabola_b;
   double parabola_c;

   int f_smooth_curve;
   std::string smooth_curve_label;
   int smooth_curve_N;
   double smooth_curve_boundary;
   double smooth_curve_t_min;
   double smooth_curve_t_max;
   polish::function_polish_description *FP_descr;


   int f_make_table_of_surfaces;

   int f_create_surface_reports;
   std::string create_surface_reports_field_orders_text;

   int f_create_surface_atlas;
   int create_surface_atlas_q_max;

   int f_create_dickson_atlas;
```

```
public:


   interface_projective();
   void print_help(int argc,
         std::string *argv, int i, int verbose_level);
   int recognize_keyword(int argc,
         std::string *argv, int i, int verbose_level);
   void read_arguments(int argc,
         std::string *argv, int &i, int verbose_level);
   void print();
   void worker(int verbose_level);

};

// ##############################################################################
// interface_symbol_table.cpp
// ##############################################################################

//! interface to the orbiter symbol table


class interface_symbol_table {

public:
   orbiter_top_level_session *Orbiter_top_level_session;

   int f_define;
   symbol_definition *Symbol_definition;

   int f_print_symbols;

   int f_with;
   std::vector<std::string> with_labels;

   int f_activity;
   activity_description *Activity_description;





   interface_symbol_table();
```

```
   void init(
         orbiter_top_level_session *Orbiter_top_level_session,
         int verbose_level);
   void print_help(int argc, std::string *argv, int i, int verbose_level);
   int recognize_keyword(int argc,
         std::string *argv, int i, int verbose_level);
   void read_arguments(
         int argc, std::string *argv, int &i, int verbose_level);
   void read_with(
         int argc, std::string *argv, int &i, int verbose_level);
   void worker(int verbose_level);
   void print();
   void print_with();

};



// ##############################################################################
// interface_toolkit.cpp
// ##############################################################################

//! interface to the general toolkit


class interface_toolkit {

   int f_create_files;
   orbiter_kernel_system::create_file_description
      *Create_file_description;

   int f_save_matrix_csv;
   std::string save_matrix_csv_label;

   int f_csv_file_select_rows;
   std::string csv_file_select_rows_fname;
   std::string csv_file_select_rows_text;

   int f_csv_file_split_rows_modulo;
   std::string csv_file_split_rows_modulo_fname;
   int csv_file_split_rows_modulo_n;

   int f_csv_file_select_cols;
   std::string csv_file_select_cols_fname;
   std::string csv_file_select_cols_text;

   int f_csv_file_select_rows_and_cols;
```

```
std::string csv_file_select_rows_and_cols_fname;
std::string csv_file_select_rows_and_cols_R_text;
std::string csv_file_select_rows_and_cols_C_text;

int f_csv_file_sort_each_row;
std::string csv_file_sort_each_row_fname;


int f_csv_file_join;
std::vector<std::string> csv_file_join_fname;
std::vector<std::string> csv_file_join_identifier;

int f_csv_file_concatenate;
std::string csv_file_concatenate_fname_out;
std::vector<std::string> csv_file_concatenate_fname_in;

int f_csv_file_concatenate_from_mask;
int csv_file_concatenate_from_mask_N;
std::string csv_file_concatenate_from_mask_mask;
std::string csv_file_concatenate_from_mask_fname_out;

int f_csv_file_extract_column_to_txt;
std::string csv_file_extract_column_to_txt_fname;
std::string csv_file_extract_column_to_txt_col_label;


int f_csv_file_latex;
int f_produce_latex_header;
std::string csv_file_latex_fname;

int f_grade_statistic_from_csv;
std::string grade_statistic_from_csv_fname;
std::string grade_statistic_from_csv_m1_label;
std::string grade_statistic_from_csv_m2_label;
std::string grade_statistic_from_csv_final_label;
std::string grade_statistic_from_csv_oracle_grade_label;

int f_draw_matrix;
graphics::draw_bitmap_control *Draw_bitmap_control;

int f_reformat;
std::string reformat_fname_in;
std::string reformat_fname_out;
int reformat_nb_cols;

int f_split_by_values;
std::string split_by_values_fname_in;
```

```
int f_change_values;
std::string change_values_fname_in;
std::string change_values_fname_out;
std::string change_values_function_input;
std::string change_values_function_output;

int f_store_as_csv_file;
std::string store_as_csv_file_fname;
int store_as_csv_file_m;
int store_as_csv_file_n;
std::string store_as_csv_file_data;

int f_mv;
std::string mv_a;
std::string mv_b;

int f_system;
std::string system_command;


int f_loop;
int loop_start_idx;
int loop_end_idx;
std::string loop_variable;
int loop_from;
int loop_to;
int loop_step;
std::string *loop_argv;

int f_plot_function;
std::string plot_function_fname;

int f_draw_projective_curve;
graphics::draw_projective_curve_description
   *Draw_projective_curve_description;

int f_tree_draw;
graphics::tree_draw_options *Tree_draw_options;

int f_extract_from_file;
std::string extract_from_file_fname;
std::string extract_from_file_label;
std::string extract_from_file_target_fname;

int f_extract_from_file_with_tail;
std::string extract_from_file_with_tail_fname;
```

```
   std::string extract_from_file_with_tail_label;
   std::string extract_from_file_with_tail_tail;
   std::string extract_from_file_with_tail_target_fname;

   int f_serialize_file_names;
   std::string serialize_file_names_fname;
   std::string serialize_file_names_output_mask;

public:


   interface_toolkit();
   void print_help(int argc,
         std::string *argv, int i, int verbose_level);
   int recognize_keyword(int argc,
         std::string *argv,
         int i, int verbose_level);
   void read_arguments(int argc,
         std::string *argv, int &i, int verbose_level);
   void print();
   void worker(int verbose_level);

};

// ##############################################################################
// orbiter_command.cpp
// ##############################################################################



//! a single command in the Orbiter dash code language


class orbiter_command {

public:

   orbiter_top_level_session *Orbiter_top_level_session;


   int f_algebra;
   interface_algebra *Algebra;

   int f_coding_theory;
   interface_coding_theory *Coding_theory;

   int f_combinatorics;
```

```
   interface_combinatorics *Combinatorics;

   int f_cryptography;
   interface_cryptography *Cryptography;

   int f_povray;
   interface_povray *Povray;

   int f_projective;
   interface_projective *Projective;

   int f_symbol_table;
   interface_symbol_table *Symbol_table;

   int f_toolkit;
   interface_toolkit *Toolkit;

   orbiter_command();
   ~orbiter_command();
   void parse(orbiter_top_level_session *Orbiter_top_level_session,
         int argc, std::string *Argv, int &i, int verbose_level);
   void execute(int verbose_level);
   void print();

};


// ##############################################################################
// orbiter_top_level_session.cpp
// ##############################################################################



//! The top level orbiter session is responsible for the command line interface a
nd the program execution and for the orbiter_session


class orbiter_top_level_session {

public:

   orbiter_kernel_system::orbiter_session *Orbiter_session;

   orbiter_top_level_session();
   ~orbiter_top_level_session();
   int startup_and_read_arguments(int argc,
         std::string *argv, int i0);
```

```
   void handle_everything(
         int argc, std::string *Argv, int i, int verbose_level);
   void parse_and_execute(
         int argc, std::string *Argv, int i, int verbose_level);
   void parse(
         int argc, std::string *Argv, int &i,
         std::vector<void * > &program, int verbose_level);
   void *get_object(int idx);
   symbol_table_object_type get_object_type(int idx);
   int find_symbol(std::string &label);
   void find_symbols(std::vector<std::string> &Labels, int *&Idx);
   void print_symbol_table();
   void add_symbol_table_entry(
         std::string &label,
         orbiter_kernel_system::orbiter_symbol_table_entry *Symb,
         int verbose_level);
   apps_algebra::any_group *get_object_of_type_any_group(
         std::string &label);
   projective_geometry::projective_space_with_action
      *get_object_of_type_projective_space(
         std::string &label);
   poset_classification::poset_classification_control
      *get_object_of_type_poset_classification_control(
            std::string &label);
   void get_vector_or_set(std::string &label,
         long int *&Pts, int &nb_pts, int verbose_level);
   apps_algebra::vector_ge_builder
      *get_object_of_type_vector_ge(
            std::string &label);
   orthogonal_geometry_applications::orthogonal_space_with_action
      *get_object_of_type_orthogonal_space_with_action(
            std::string &label);
   spreads::spread_create
      *get_object_of_type_spread(
            std::string &label);
   applications_in_algebraic_geometry::cubic_surfaces_in_general::surface_create
      *get_object_of_type_cubic_surface(
            std::string &label);
   apps_coding_theory::create_code
      *get_object_of_type_code(
            std::string &label);
   orthogonal_geometry_applications::orthogonal_space_with_action
      *get_orthogonal_space(
            std::string &label);

};
```

```
// ##############################################################################
// symbol_definition.cpp
// ##############################################################################

//! definition of an orbiter symbol


class symbol_definition {

public:
   interface_symbol_table *Sym;


   std::string define_label;

   int f_finite_field;
   field_theory::finite_field_description
      *Finite_field_description;

   int f_polynomial_ring;
   ring_theory::polynomial_ring_description
      *Polynomial_ring_description;

   int f_projective_space;
   projective_geometry::projective_space_with_action_description
      *Projective_space_with_action_description;

   int f_orthogonal_space;
   orthogonal_geometry_applications::orthogonal_space_with_action_description
      *Orthogonal_space_with_action_description;

   int f_BLT_set_classifier;
   std::string BLT_set_classifier_label_orthogonal_geometry;
   orthogonal_geometry_applications::blt_set_classify_description
      *Blt_set_classify_description;

   int f_spread_classifier;
   spreads::spread_classify_description
      *Spread_classify_description;

   int f_linear_group;
   groups::linear_group_description
      *Linear_group_description;

   int f_permutation_group;
   groups::permutation_group_description
```

```
      *Permutation_group_description;

   int f_group_modification;
   apps_algebra::group_modification_description
      *Group_modification_description;

   int f_formula;
   expression_parser::formula *Formula;
   std::string label;
   std::string label_tex;
   std::string managed_variables;
   std::string formula_text;

   int f_collection;
   std::string list_of_objects;

   int f_geometric_object;
   std::string geometric_object_projective_space_label;
   geometry::geometric_object_description
      *Geometric_object_description;

   int f_graph;
   apps_graph_theory::create_graph_description
      *Create_graph_description;

   int f_code;
   apps_coding_theory::create_code_description
      *Create_code_description;

   int f_spread;
   spreads::spread_create_description
      *Spread_create_description;

   int f_cubic_surface;
   applications_in_algebraic_geometry::cubic_surfaces_in_general::surface_create_d
escription
      *Surface_Descr;

   int f_quartic_curve;
   applications_in_algebraic_geometry::quartic_curves::quartic_curve_create_descri
ption
      *Quartic_curve_descr;

   int f_BLT_set;
   orthogonal_geometry_applications::BLT_set_create_description
      *BLT_Set_create_description;
```

```
int f_translation_plane;
std::string translation_plane_spread_label;
std::string translation_plane_group_n_label;
std::string translation_plane_group_np1_label;


int f_spread_table;
std::string spread_table_label_PA;
int dimension_of_spread_elements;
std::string spread_selection_text;
std::string spread_tables_prefix;

int f_packing_was;
std::string packing_was_label_spread_table;
packings::packing_was_description * packing_was_descr;

int f_packing_was_choose_fixed_points;
std::string packing_with_assumed_symmetry_label;
int packing_with_assumed_symmetry_choose_fixed_points_clique_size;
poset_classification::poset_classification_control
   *packing_with_assumed_symmetry_choose_fixed_points_control;


int f_packing_long_orbits;
std::string packing_long_orbits_choose_fixed_points_label;
packings::packing_long_orbits_description
   * Packing_long_orbits_description;

int f_graph_classification;
apps_graph_theory::graph_classify_description
   * Graph_classify_description;

int f_diophant;
solvers::diophant_description
   *Diophant_description;

int f_design;
apps_combinatorics::design_create_description
   *Design_create_description;

int f_design_table;
std::string design_table_label_design;
std::string design_table_label;
std::string design_table_group;


int f_large_set_was;
```

```
std::string  large_set_was_label_design_table;
apps_combinatorics::large_set_was_description
   *large_set_was_descr;

int f_set;
data_structures::set_builder_description
   *Set_builder_description;

int f_vector;
data_structures::vector_builder_description
   *Vector_builder_description;

int f_combinatorial_objects;
data_structures::data_input_stream_description
   *Data_input_stream_description;

int f_geometry_builder;
geometry_builder::geometry_builder_description
   *Geometry_builder_description;

int f_vector_ge;
data_structures_groups::vector_ge_description
   *Vector_ge_description;

int f_action_on_forms;
apps_algebra::action_on_forms_description
   *Action_on_forms_descr;

int f_orbits;
apps_algebra::orbits_create_description
   *Orbits_create_description;

int f_poset_classification_control;
poset_classification::poset_classification_control
   *Poset_classification_control;

symbol_definition();
~symbol_definition();
void read_definition(
      interface_symbol_table *Sym,
      int argc, std::string *argv, int &i, int verbose_level);
void perform_definition(int verbose_level);
void print();
void definition_of_finite_field(int verbose_level);
void definition_of_polynomial_ring(int verbose_level);
void definition_of_projective_space(int verbose_level);
void print_definition_of_projective_space(int verbose_level);
```

```
   void definition_of_orthogonal_space(int verbose_level);
   void definition_of_BLT_set_classifier(int verbose_level);
   void definition_of_spread_classifier(int verbose_level);
   void definition_of_linear_group(int verbose_level);
   void definition_of_permutation_group(int verbose_level);
   void definition_of_modified_group(int verbose_level);
   void definition_of_geometric_object(int verbose_level);
   void definition_of_formula(
         expression_parser::formula *Formula,
         int verbose_level);
   void definition_of_collection(std::string &list_of_objects,
         int verbose_level);
   void definition_of_graph(int verbose_level);
   void definition_of_code(int verbose_level);
   void definition_of_spread(int verbose_level);
   void definition_of_cubic_surface(int verbose_level);
   void definition_of_quartic_curve(int verbose_level);
   void definition_of_BLT_set(int verbose_level);
   void definition_of_translation_plane(int verbose_level);
   void definition_of_spread_table(int verbose_level);
   void definition_of_packing_was(int verbose_level);
   void definition_of_packing_was_choose_fixed_points(
         int verbose_level);
   void definition_of_packing_long_orbits(int verbose_level);
   void definition_of_graph_classification(int verbose_level);
   void definition_of_diophant(int verbose_level);
   void definition_of_design(int verbose_level);
   void definition_of_design_table(int verbose_level);
   void definition_of_large_set_was(int verbose_level);
   void definition_of_set(int verbose_level);
   void definition_of_vector(
         std::string &label,
         data_structures::vector_builder_description *Descr,
         int verbose_level);
   void definition_of_combinatorial_object(int verbose_level);
   void do_geometry_builder(int verbose_level);
   void load_finite_field_PG(int verbose_level);
   void load_finite_field(std::string &input_q,
         field_theory::finite_field *&F, int verbose_level);
   void definition_of_vector_ge(int verbose_level);
   void definition_of_action_on_forms(int verbose_level);
   void definition_of_orbits(int verbose_level);
   void definition_of_poset_classification_control(int verbose_level);

};
```

```
// ##############################################################################
// global variable:
// ##############################################################################



extern user_interface::orbiter_top_level_session
   *The_Orbiter_top_level_session;
   // global top level Orbiter session



}}}


#endif /* SRC_LIB_TOP_LEVEL_INTERFACES_INTERFACES_H_ */
```

7.8 Orthogonal Geometry

```
/*
 * tl_orthogonal.h
 *
 *  Created on: Jun 9, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_ORTHOGONAL_TL_ORTHOGONAL_H_
#define SRC_LIB_TOP_LEVEL_ORTHOGONAL_TL_ORTHOGONAL_H_


namespace orbiter {
namespace layer5_applications {
namespace orthogonal_geometry_applications {



// ##############################################################################
// blt_set_activity_description.cpp
// ##############################################################################

//! description of an activity for a BLT-set



class blt_set_activity_description {

public:

   int f_report;

   int f_create_flock;
   int create_flock_point_idx;

   blt_set_activity_description();
   ~blt_set_activity_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// blt_set_activity.cpp
```

```
// ##############################################################################

//! an activity regarding a BLT-sets



class blt_set_activity {

public:

   blt_set_activity_description *Descr;

   BLT_set_create *BC;

   blt_set_activity();
   ~blt_set_activity();
   void init(blt_set_activity_description *Descr,
         orthogonal_geometry_applications::BLT_set_create *BC,
         int verbose_level);
   void perform_activity(int verbose_level);

};



// ##############################################################################
// blt_set_classify_activity_description.cpp
// ##############################################################################

//! description of an activity regarding the classification of BLT-sets



class blt_set_classify_activity_description {

public:

   int f_compute_starter;
   poset_classification::poset_classification_control *starter_control;

   int f_create_graphs;

   int f_split;
   int split_r;
   int split_m;

   int f_isomorph;
```

```
   layer4_classification::isomorph::isomorph_arguments
      *Isomorph_arguments;

   blt_set_classify_activity_description();
   ~blt_set_classify_activity_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// blt_set_classify_activity.cpp
// ##############################################################################

//! an activity regarding the classification of BLT-sets



class blt_set_classify_activity {

public:

   blt_set_classify_activity_description *Descr;
   blt_set_classify *BLT_classify;
   orthogonal_space_with_action *OA;

   blt_set_classify_activity();
   ~blt_set_classify_activity();
   void init(blt_set_classify_activity_description *Descr,
         blt_set_classify *BLT_classify,
         orthogonal_space_with_action *OA,
         int verbose_level);
   void perform_activity(int verbose_level);

};



// ##############################################################################
// blt_set_classify_description.cpp
// ##############################################################################

//! classification of BLT-sets
```

```
class blt_set_classify_description {

public:

   int f_starter_size;
   int starter_size;

   blt_set_classify_description();
   ~blt_set_classify_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// blt_set_classify.cpp
// ##############################################################################

//! classification of BLT-sets



class blt_set_classify {

public:

   orthogonal_space_with_action *OA;

   layer1_foundations::orthogonal_geometry::blt_set_domain
      *Blt_set_domain;

   actions::action *A; // orthogonal group



   int starter_size;

   groups::strong_generators *Strong_gens;

   int f_semilinear;
```

```
int q;

poset_classification::poset_classification_control *Control;
poset_classification::poset_with_group_action *Poset;
poset_classification::poset_classification *gen;
int degree;


int target_size;

isomorph::isomorph_worker *Worker;


blt_set_classify();
~blt_set_classify();
void init_basic(orthogonal_space_with_action *OA,
      actions::action *A,
      groups::strong_generators *Strong_gens,
      int starter_size,
      int verbose_level);
void compute_starter(
      poset_classification::poset_classification_control *Control,
      int verbose_level);
void create_graphs(
   int orbit_at_level_r, int orbit_at_level_m,
   int level_of_candidates_file,
   int f_lexorder_test, int f_eliminate_graphs_if_possible,
   int verbose_level);
void create_graphs_list_of_cases(
      std::string &case_label,
      std::string &list_of_cases_text,
   int level_of_candidates_file,
   int f_lexorder_test, int f_eliminate_graphs_if_possible,
   int verbose_level);
int create_graph(
   int orbit_at_level, int level_of_candidates_file,
   int f_lexorder_test, int f_eliminate_graphs_if_possible,
   int &nb_vertices,
   graph_theory::colored_graph *&CG,
   int verbose_level);

void lifting_prepare_function_new(
      solvers_package::exact_cover *E, int starter_case,
   long int *candidates, int nb_candidates,
   groups::strong_generators *Strong_gens,
   solvers::diophant *&Dio, long int *&col_labels,
   int &f_ruled_out,
```

```
      int verbose_level);
   void report_from_iso(isomorph::isomorph &Iso, int verbose_level);
   void report(data_structures_groups::orbit_transversal *T,
         int verbose_level);
   void report2(std::ostream &ost,
         data_structures_groups::orbit_transversal *T,
         int verbose_level);
};



// ##############################################################################
// BLT_set_create_description.cpp
// ##############################################################################

//! to create BLT set with a description from the command line



class BLT_set_create_description {

public:

   int f_catalogue;
   int iso;
   int f_family;
   std::string family_name;

   int f_space;
   std::string space_label;



   BLT_set_create_description();
   ~BLT_set_create_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();
};



// ##############################################################################
// BLT_set_create.cpp
// ##############################################################################
```

```
//! to create a BLT-set from a description using class BLT_set_create_description




class BLT_set_create {

public:
   BLT_set_create_description *Descr;

   std::string prefix;
   std::string label_txt;
   std::string label_tex;



   orthogonal_space_with_action *OA;

   long int *set;

   int *ABC;


   int f_has_group;
   groups::strong_generators *Sg;

   layer1_foundations::orthogonal_geometry::blt_set_domain
      *Blt_set_domain;
   blt_set_with_action *BA;


   BLT_set_create();
   ~BLT_set_create();
   void init(
         layer1_foundations::orthogonal_geometry::blt_set_domain
            *Blt_set_domain,
         BLT_set_create_description *Descr,
         orthogonal_space_with_action *OA,
         int verbose_level);
   void apply_transformations(
         std::vector<std::string> transform_coeffs,
         std::vector<int> f_inverse_transform, int verbose_level);
   void report(int verbose_level);
   void create_flock(int point_idx, int verbose_level);
   void report2(std::ostream &ost, int verbose_level);
   void print_set_of_points(
            std::ostream &ost, long int *Pts, int nb_pts);
```

```
   void print_set_of_points_with_ABC(
            std::ostream &ost, long int *Pts, int nb_pts);

};



// ##############################################################################
// blt_set_with_action.cpp
// ##############################################################################


//! a BLT-set together with its stabilizer


class blt_set_with_action {

public:

   actions::action *A;
   orthogonal_geometry::blt_set_domain *Blt_set_domain;

   long int *set;

   groups::strong_generators *Aut_gens;
   orthogonal_geometry::blt_set_invariants *Inv;

   actions::action *A_on_points;
   groups::schreier *Orbits_on_points;

   long int *T; // [target_size]
   long int *Pi_ij; // [target_size * target_size]

   blt_set_with_action();
   ~blt_set_with_action();
   void init_set(
         actions::action *A,
         orthogonal_geometry::blt_set_domain *Blt_set_domain,
         long int *set,
         groups::strong_generators *Aut_gens, int verbose_level);
   void init_orbits_on_points(
         int verbose_level);
   void print_automorphism_group(
      std::ostream &ost);
   void report(std::ostream &ost, int verbose_level);
   void compute_T(int verbose_level);
   void compute_Pi_ij(int verbose_level);
```

```
};


// ##############################################################################
// flock.cpp
// ##############################################################################


//! a flock of a quadratic cone


class flock {

public:

   blt_set_with_action *BLT_set;

   int point_idx;

   long int *Flock; // [q]
   long int *Flock_reduced; // [q]
   long int *Flock_affine; // [q]

   int *ABC; // [q * 3]

   data_structures::int_matrix *Table_of_ABC;
      // same as ABC, but sorted by rows.

   int *func_f; // second column of ABC
   int *func_g; // third column of ABC

   combinatorics::polynomial_function_domain *PF;

   flock();
   ~flock();
   void init(
            blt_set_with_action *BLT_set,
            int point_idx, int verbose_level);
   void test_flock_condition(
         field_theory::finite_field *F,
         int f_magic, int *ABC, int verbose_level);
   void quadratic_lift(
         int *coeff_f, int *coeff_g, int verbose_level);
   void cubic_lift(
         int *coeff_f, int *coeff_g, int verbose_level);
```

```
};


// ##############################################################################
// orthogonal_space_activity_description.cpp
// ##############################################################################

//! description of an activity associated with an orthogonal space


class orthogonal_space_activity_description {
public:


   int f_create_BLT_set;
   BLT_set_create_description * BLT_Set_create_description;

   int f_cheat_sheet_orthogonal;

   int f_print_points;
   std::string print_points_label;

   int f_print_lines;
   std::string print_lines_label;

   int f_unrank_line_through_two_points;
   std::string unrank_line_through_two_points_p1;
   std::string unrank_line_through_two_points_p2;

   int f_lines_on_point;
   long int lines_on_point_rank;

   int f_perp;
   std::string perp_text;

   int f_set_stabilizer;
   int set_stabilizer_intermediate_set_size;
   std::string set_stabilizer_fname_mask;
   int set_stabilizer_nb;
   std::string set_stabilizer_column_label;
   std::string set_stabilizer_fname_out;

   int f_export_point_line_incidence_matrix;

   int f_intersect_with_subspace;
   std::string intersect_with_subspace_label;
```

```
   orthogonal_space_activity_description();
   ~orthogonal_space_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();


};

// ##############################################################################
// orthogonal_space_activity.cpp
// ##############################################################################

//! an activity associated with an orthogonal space


class orthogonal_space_activity {
public:

   orthogonal_space_activity_description *Descr;

   orthogonal_space_with_action *OA;

   orthogonal_space_activity();
   ~orthogonal_space_activity();
   void init(orthogonal_space_activity_description *Descr,
         orthogonal_space_with_action *OA,
         int verbose_level);
   void perform_activity(int verbose_level);
   void set_stabilizer(
         orthogonal_space_with_action *OA,
         int intermediate_subset_size,
         std::string &fname_mask, int nb, std::string &column_label,
         std::string &fname_out,
         int verbose_level);


};


// ##############################################################################
// orthogonal_space_with_action_description.cpp
// ##############################################################################
```

```
//! description of an orthogonal space with action

class orthogonal_space_with_action_description {
public:

   int epsilon;

   int n;

   std::string input_q;

   field_theory::finite_field *F;

   int f_label_txt;
   std::string label_txt;
   int f_label_tex;
   std::string label_tex;
   int f_without_group;

   orthogonal_space_with_action_description();
   ~orthogonal_space_with_action_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// orthogonal_space_with_action.cpp
// ##############################################################################


//! an orthogonal space with action

class orthogonal_space_with_action {
public:

   orthogonal_space_with_action_description *Descr;

   std::string label_txt;
   std::string label_tex;
```

```
   layer1_foundations::orthogonal_geometry::orthogonal *O;

   int f_semilinear;

   actions::action *A;
   induced_actions::action_on_orthogonal *AO;

   orthogonal_geometry::blt_set_domain *Blt_Set_domain;


   orthogonal_space_with_action();
   ~orthogonal_space_with_action();
   void init(
         orthogonal_space_with_action_description *Descr,
         int verbose_level);
   void init_group(int verbose_level);
   void report(
         graphics::layered_graph_draw_options *LG_Draw_options,
         int verbose_level);
   void report2(
         std::ostream &ost,
         graphics::layered_graph_draw_options *LG_Draw_options,
         int verbose_level);
   void report_point_set(
         long int *Pts, int nb_pts,
         std::string &label_txt,
         int verbose_level);
   void report_line_set(
         long int *Lines, int nb_lines,
         std::string &label_txt,
         int verbose_level);

};





}}}



#endif /* SRC_LIB_TOP_LEVEL_ORTHOGONAL_TL_ORTHOGONAL_H_ */
```

7.9 Packings

```
/*
 * spreads_and_packings.h
 *
 *  Created on: Jul 27, 2020
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_PACKINGS_PACKINGS_H_
#define SRC_LIB_TOP_LEVEL_PACKINGS_PACKINGS_H_


namespace orbiter {
namespace layer5_applications {
namespace packings {


// ##############################################################################
// invariants_packing.cpp
// ##############################################################################

//! collection of invariants of a set of packings in PG(3,q)

class invariants_packing {
public:
   spreads::spread_classify *T;
   packing_classify *P;
   isomorph::isomorph *Iso;
      // the classification of packings


   packing_invariants *Inv;
   ring_theory::longinteger_object *Ago, *Ago_induced;
   int *Ago_int;

   int *Spread_type_of_packing;
      // [Iso->Reps->count * P->nb_iso_types_of_spreads]


   data_structures::tally_vector_data *Classify;


   int *Dual_idx;
      // [Iso->Reps->count]
   int *f_self_dual;
      // [Iso->Reps->count]
```

```
   invariants_packing();
   ~invariants_packing();
   void init(
         isomorph::isomorph *Iso,
         packing_classify *P, int verbose_level);
   void compute_dual_packings(
         isomorph::isomorph *Iso, int verbose_level);
   void make_table(
         isomorph::isomorph *Iso, std::ostream &ost,
      int f_only_self_dual,
      int f_only_not_self_dual,
      int verbose_level);
};


// ##############################################################################
// packing_classify.cpp
// ##############################################################################

//! classification of packings in PG(3,q)

class packing_classify {
public:

   projective_geometry::projective_space_with_action *PA;

   std::string path_to_spread_tables;

   spreads::spread_classify *T;
   field_theory::finite_field *F;
   int spread_size;
   int nb_lines;


   int f_lexorder_test;
   int q;
   int size_of_packing;
      // the number of spreads in a packing,
      // which is q^2 + q + 1

   spreads::spread_table_with_selection *Spread_table_with_selection;

   geometry::projective_space *P3;
   geometry::projective_space *P5;
   long int *the_packing; // [size_of_packing]
   long int *spread_iso_type; // [size_of_packing]
   long int *dual_packing; // [size_of_packing]
```

```
long int *list_of_lines; // [size_of_packing * spread_size]
long int *list_of_lines_klein_image; // [size_of_packing * spread_size]
geometry::grassmann *Gr; // the Grassmannian Gr_{6,3}



int *degree;

poset_classification::poset_classification_control *Control;
poset_classification::poset_with_group_action *Poset;
poset_classification::poset_classification *gen;

int nb_needed;


packing_classify();
~packing_classify();
void spread_table_init(
      projective_geometry::projective_space_with_action *PA,
      int dimension_of_spread_elements,
      int f_select_spread, std::string &select_spread_text,
      std::string &path_to_spread_tables,
      int verbose_level);
void init(
      projective_geometry::projective_space_with_action *PA,
      spreads::spread_table_with_selection
         *Spread_table_with_selection,
      int f_lexorder_test,
      int verbose_level);
void init2(
      poset_classification::poset_classification_control *Control,
      int verbose_level);
void init_P3_and_P5_and_Gr(int verbose_level);
void compute_adjacency_matrix(int verbose_level);
void prepare_generator(
      poset_classification::poset_classification_control *Control,
      int verbose_level);
void compute(int search_depth, int verbose_level);
void lifting_prepare_function_new(
      solvers_package::exact_cover *E,
      int starter_case,
   long int *candidates, int nb_candidates,
   groups::strong_generators *Strong_gens,
   solvers::diophant *&Dio, long int *&col_labels,
   int &f_ruled_out,
   int verbose_level);
void report_fixed_objects(
```

```
      int *Elt, char *fname_latex,
      int verbose_level);
int test_if_orbit_is_partial_packing(
      groups::schreier *Orbits, int orbit_idx,
   long int *orbit1, int verbose_level);
int test_if_pair_of_orbits_are_adjacent(
      groups::schreier *Orbits, int a, int b,
   long int *orbit1, long int *orbit2, int verbose_level);
// tests if every spread from orbit a
// is line-disjoint from every spread from orbit b
int find_spread(long int *set, int verbose_level);

// packing2.cpp
void compute_klein_invariants(
      isomorph::isomorph *Iso,
      int f_split, int split_r, int split_m,
      int verbose_level);
void klein_invariants_fname(
      std::string &fname,
      std::string &prefix, int iso_cnt);
void compute_and_save_klein_invariants(
      std::string &prefix,
   int iso_cnt,
   long int *data, int data_size, int verbose_level);
void report(
      isomorph::isomorph *Iso, int verbose_level);
void report_whole(
      isomorph::isomorph *Iso,
      std::ostream &ost, int verbose_level);
void report_title_page(
      isomorph::isomorph *Iso,
      std::ostream &ost, int verbose_level);
void report_packings_by_ago(
      isomorph::isomorph *Iso,
      std::ostream &ost,
   invariants_packing *inv,
   data_structures::tally &C_ago,
   int verbose_level);
void report_isomorphism_type(
      isomorph::isomorph *Iso, std::ostream &ost,
   int orbit, invariants_packing *inv, int verbose_level);
void report_packing_as_table(
      isomorph::isomorph *Iso, std::ostream &ost,
   int orbit, invariants_packing *inv, long int *list_of_lines,
   int verbose_level);
void report_klein_invariants(
      isomorph::isomorph *Iso, std::ostream &ost,
```

```
      int orbit, invariants_packing *inv, int verbose_level);
   void report_stabilizer(
         isomorph::isomorph &Iso, std::ostream &ost, int orbit,
         int verbose_level);
   void report_stabilizer_in_action(
         isomorph::isomorph &Iso,
         std::ostream &ost, int orbit, int verbose_level);
   void report_stabilizer_in_action_gap(
         isomorph::isomorph &Iso,
         int orbit, int verbose_level);
   void report_extra_stuff(
         isomorph::isomorph *Iso, std::ostream &ost,
         int verbose_level);
};



// ##############################################################################
// packing_invariants.cpp
// ##############################################################################

//! geometric invariants of a packing in PG(3,q)

class packing_invariants {
public:
   packing_classify *P;

   std::string prefix;
   std::string prefix_tex;
   int iso_cnt;

   long int *the_packing;
      // [P->size_of_packing]

   long int *list_of_lines;
      // [P->size_of_packing * P->spread_size]

   int f_has_klein;
   ring_theory::longinteger_object *R;
   int **Pts_on_plane;
   int *nb_pts_on_plane;
   int nb_planes;

   data_structures::tally *C;
   int nb_blocks;
   int *block_to_plane; // [nb_blocks]
   int *plane_to_block; // [nb_planes]
```

```
   int nb_fake_blocks;
   int nb_fake_points;
   int total_nb_blocks;
      // nb_blocks + nb_fake_blocks
   int total_nb_points;
      // P->size_of_packing * P->spread_size + nb_fake_points
   int *Inc;
      // [total_nb_points * total_nb_blocks]

   geometry::incidence_structure *I;
   data_structures::partitionstack *Stack;
   std::string fname_incidence_pic;
   std::string fname_row_scheme;
   std::string fname_col_scheme;

   packing_invariants();
   ~packing_invariants();
   void init(packing_classify *P,
         std::string &prefix,
         std::string &prefix_tex,
         int iso_cnt,
      long int *the_packing, int verbose_level);
   void init_klein_invariants(typed_objects::Vector &v, int verbose_level);
   void compute_decomposition(int verbose_level);
};

// ##############################################################################
// packing_long_orbits_description.cpp
// ##############################################################################

//! command line description of picking long orbits of packings with assumed symm
etry

class packing_long_orbits_description {
public:
   int f_split;
   int split_r;
   int split_m;

   int f_orbit_length;
   int orbit_length;

   int f_mixed_orbits;
   std::string mixed_orbits_length_text;

   int f_list_of_cases_from_file;
   std::string list_of_cases_from_file_fname;
```

```
   int f_solution_path;
   std::string solution_path;

   int f_create_graphs;

   int f_solve;

   int f_read_solutions;


   packing_long_orbits_description();
   ~packing_long_orbits_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};

// ##############################################################################
// packing_long_orbits.cpp
// ##############################################################################

//! complete a partial packing from a clique on the fixpoint graph using long orb
its, utilizing clique search

class packing_long_orbits {
public:
   packing_was_fixpoints *PWF;
   packing_long_orbits_description *Descr;

   int fixpoints_idx;
   int fixpoint_clique_size;

   int *Orbit_lengths;
   int nb_orbit_lengths;
   int *Type_idx;

   int long_orbit_idx;
   long int *set; // [Descr->orbit_length]




   int fixpoints_clique_case_number;
   long int *fixpoint_clique_orbit_numbers;
   groups::strong_generators *fixpoint_clique_stabilizer_gens;
```

```
   long int *fixpoint_clique;
   data_structures::set_of_sets *Filtered_orbits;

   std::string fname_graph;
   std::string fname_solutions;


   packing_long_orbits();
   ~packing_long_orbits();
   void init(packing_was_fixpoints *PWF,
         packing_long_orbits_description *Descr,
         int verbose_level);
   void list_of_cases_from_file(int verbose_level);
   void save_packings_by_case(
         std::string &fname_packings,
         std::vector<std::vector<std::vector<int> > >
            &Packings_by_case,
         int verbose_level);
   void process_single_case(
         std::vector<std::vector<int> > &Packings_classified,
         std::vector<std::vector<int> > &Packings,
         int verbose_level);
   void init_fixpoint_clique_from_orbit_numbers(int verbose_level);
   void filter_orbits(int verbose_level);
   void create_graph_on_remaining_long_orbits(
         std::vector<std::vector<int> > &Packings_classified,
         std::vector<std::vector<int> > &Packings,
         int verbose_level);
   void create_fname_graph_on_remaining_long_orbits();
   void create_graph_and_save_to_file(
         graph_theory::colored_graph *&CG,
         std::string &fname,
         int f_has_user_data,
         long int *user_data, int user_data_size,
         int verbose_level);
   void create_graph_on_long_orbits(
         graph_theory::colored_graph *&CG,
         long int *user_data, int user_data_sz,
         int verbose_level);
   void report_filtered_orbits(std::ostream &ost);

};



// ##############################################################################
// packing_was_activity_description.cpp
```

```
// ##############################################################################

//! description of an activity involving a packing_was

class packing_was_activity_description {
public:

   int f_report;

   int f_export_reduced_spread_orbits;
   std::string export_reduced_spread_orbits_fname_base;

   int f_create_graph_on_mixed_orbits;
   std::string create_graph_on_mixed_orbits_orbit_lengths;


   packing_was_activity_description();
   ~packing_was_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// packing_was_activity.cpp
// ##############################################################################

//! an activity involving a packing_was

class packing_was_activity {
public:

   packing_was_activity_description *Descr;
   packing_was *PW;

   packing_was_activity();
   ~packing_was_activity();
   void init(packing_was_activity_description *Descr,
         packing_was *PW,
         int verbose_level);
   void perform_activity(int verbose_level);
};
```

```
// ##############################################################################
// packing_was_description.cpp
// ##############################################################################

//! command line description of tasks for packings with assumed symmetry

class packing_was_description {
public:

   int f_process_long_orbits;
   packing_long_orbits_description *Long_Orbits_Descr;

   int f_fixp_clique_types_save_individually;

   int f_spread_tables_prefix;
   std::string spread_tables_prefix;

   int f_exact_cover;
   solvers_package::exact_cover_arguments *ECA;

   int f_isomorph;
   isomorph::isomorph_arguments *IA;

   int f_H;
   std::string H_label;
   groups::linear_group_description *H_Descr;

   int f_N;
   std::string N_label;
   groups::linear_group_description *N_Descr;

   int f_report;

   int f_regular_packing;

   packing_was_description();
   ~packing_was_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};

// ##############################################################################
// packing_was_fixpoints_activity_description.cpp
// ##############################################################################
```

```
//! description of an activity after the fixed points have been selected in the c
onstruction of packings in PG(3,q) with assumed symmetry

class packing_was_fixpoints_activity_description {
public:
   int f_report;

   int f_print_packing;
   std::string print_packing_text;

   int f_compare_files_of_packings;
   std::string compare_files_of_packings_fname1;
   std::string compare_files_of_packings_fname2;

   packing_was_fixpoints_activity_description();
   ~packing_was_fixpoints_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};

// ##############################################################################
// packing_was_fixpoints_activity.cpp
// ##############################################################################

//! an activity after the fixed points have been selected in the construction of
packings in PG(3,q) with assumed symmetry

class packing_was_fixpoints_activity {
public:

   packing_was_fixpoints_activity_description *Descr;
   packing_was_fixpoints *PWF;

   packing_was_fixpoints_activity();
   ~packing_was_fixpoints_activity();
   void init(
         packing_was_fixpoints_activity_description *Descr,
         packing_was_fixpoints *PWF,
         int verbose_level);
   void perform_activity(int verbose_level);

};
```

```
// ##############################################################################
// packing_was_fixpoints.cpp
// ##############################################################################

//! picking fixed points in the construction of packings in PG(3,q) with assumed
symmetry

class packing_was_fixpoints {
public:
   packing_was *PW;

   std::string fname_fixp_graph;
   std::string fname_fixp_graph_cliques;
   int fixpoints_idx;
      // index of orbits of length 1
      // in reduced_spread_orbits_under_H
   actions::action *A_on_fixpoints;
      // A_on_reduced_spread_orbits->
      // create_induced_action_by_restriction(
      // reduced_spread_orbits_under_H->
      // Orbits_classified->Set_size[fixpoints_idx],
      // reduced_spread_orbits_under_H->
      // Orbits_classified->Sets[fixpoints_idx])

   graph_theory::colored_graph *fixpoint_graph;
   poset_classification::poset_with_group_action
      *Poset_fixpoint_cliques;
   poset_classification::poset_classification
      *fixpoint_clique_gen;

   int fixpoint_clique_size;
   long int *Cliques;
      // [nb_cliques * fixpoint_clique_size]
   int nb_cliques;
   std::string fname_fixpoint_cliques_orbiter;
   data_structures_groups::orbit_transversal *Fixp_cliques;




   packing_was_fixpoints();
   ~packing_was_fixpoints();
   void init(
         packing_was *PW,
         int fixpoint_clique_size,
```

```
        poset_classification::poset_classification_control
           *Control,
        int verbose_level);
   void setup_file_names(int clique_size, int verbose_level);
   void create_graph_on_fixpoints(int verbose_level);
   void action_on_fixpoints(int verbose_level);
   void compute_cliques_on_fixpoint_graph(
        int clique_size,
        poset_classification::poset_classification_control
           *Control,
        int verbose_level);
   // initializes the orbit transversal Fixp_cliques
   // initializes Cliques[nb_cliques * clique_size]
   // (either by computing it or reading it from file)
   void compute_cliques_on_fixpoint_graph_from_scratch(
        int clique_size,
        poset_classification::poset_classification_control
           *Control,
        int verbose_level);
   // compute cliques on fixpoint graph using A_on_fixpoints
   // orbit representatives will be stored
   // in Cliques[nb_cliques * clique_size]
   void process_long_orbits(int verbose_level);
   long int *clique_by_index(int idx);
   groups::strong_generators *get_stabilizer(int idx);
   void print_packing(
        long int *packing, int sz, int verbose_level);
   void process_long_orbits(
        int clique_index,
        int f_solution_path,
        std::string &solution_path,
        std::vector<std::vector<int> > &Packings,
        int verbose_level);
   void report(int verbose_level);
   void report2(
        std::ostream &ost, int verbose_level);
   long int fixpoint_to_reduced_spread(int a, int verbose_level);

};


// ##############################################################################
// packing_was.cpp
// ##############################################################################

//! construction of packings in PG(3,q) with assumed symmetry
```

```
class packing_was {
public:
   packing_was_description *Descr;

   groups::linear_group *H_LG;

   groups::linear_group *N_LG;

   packing_classify *P;

   groups::strong_generators *H_gens;
   ring_theory::longinteger_object H_go;
   long int H_goi;
   groups::sims *H_sims;

   actions::action *A;
   int f_semilinear;
   groups::matrix_group *M;
   int dim;

   groups::strong_generators *N_gens;
   ring_theory::longinteger_object N_go;
   long int N_goi;


   std::string prefix_point_orbits_under_H;
   groups::orbits_on_something *Point_orbits_under_H;
      // using H_gens in action P->T->A


   std::string prefix_point_orbits_under_N;
   groups::orbits_on_something *Point_orbits_under_N;
      // using N_gens in action P->T->A


   std::string prefix_line_orbits_under_H;
   groups::orbits_on_something *Line_orbits_under_H;
      // using H_gens in action P->T->A2

   std::string prefix_line_orbits_under_N;
   groups::orbits_on_something *Line_orbits_under_N;
      // using H_gens in action P->T->A2

   std::string prefix_spread_types;
   data_structures_groups::orbit_type_repository *Spread_type;

   std::string prefix_spread_orbits;
```

```
groups::orbits_on_something *Spread_orbits_under_H;
   // using H_gens in action
   // P->Spread_table_with_selection->A_on_spreads


actions::action *A_on_spread_orbits;
   // derived from P->Spread_table_with_selection
   // ->A_on_spreads
   // restricted action on Spread_orbits_under_H:
   // = induced_action_on_orbits(
   // P->A_on_spreads, Spread_orbits_under_H)

std::string fname_good_orbits;
int nb_good_orbits;
long int *Good_orbit_idx;
long int *Good_orbit_len;
long int *orb;

int nb_good_spreads;
int *good_spreads;
   // the union of all good orbits on spreads

geometry::spread_tables *Spread_tables_reduced;
   // The spreads in the good orbits, listed one-by-one
   // This table is *not* sorted.
   // The induced action on reduced spreads
   // (A_on_reduced_spreads)
   // maintains a sorted table.


std::string prefix_spread_types_reduced;
data_structures_groups::orbit_type_repository
   *Spread_type_reduced;

actions::action *A_on_reduced_spreads;
   // induced action on Spread_tables_reduced

std::string prefix_reduced_spread_orbits;
groups::orbits_on_something
   *reduced_spread_orbits_under_H;
   // = reduced_spread_orbits_under_H->init(
   // A_on_reduced_spreads, H_gens)

actions::action *A_on_reduced_spread_orbits;
   // induced_action_on_orbits(A_on_reduced_spreads,
   // reduced_spread_orbits_under_H)
```

```
data_structures::set_of_sets *Orbit_invariant;
   // the values of Spread_type_reduced->type[spread_idx]
   // for the spreads in one orbit.
   // Since it is an orbit invariant,
   // the value is constant for all elements of the orbit,
   // so it need to be stored only once for each orbit.
   // more precisely, Orbit_invariant->Sets[i][j] is
   // the type of the spreads belonging to the orbit
   // reduced_spread_orbits_under_H->Orbits_classified->Sets[i][j]

int nb_sets;
data_structures::tally
   *Classify_spread_invariant_by_orbit_length;

regular_packing *Regular_packing;
   // correspondence between regular spreads
   // and external lines of the Klein quadric


packing_was();
~packing_was();
void init(packing_was_description *Descr,
      packing_classify *P, int verbose_level);
void compute_H_orbits_and_reduce(int verbose_level);
void init_regular_packing(int verbose_level);
void init_N(int verbose_level);
void init_H(int verbose_level);
void compute_H_orbits_on_points(int verbose_level);
// computes the orbits of H on points
// and writes to file prefix_point_orbits
void compute_N_orbits_on_points(int verbose_level);
// computes the orbits of N on points
// and writes to file prefix_point_orbits
void compute_H_orbits_on_lines(int verbose_level);
   // computes the orbits of H on lines (NOT on spreads!)
   // and writes to file prefix_line_orbits
void compute_N_orbits_on_lines(int verbose_level);
// computes the orbits of N on lines (NOT on spreads!)
// and writes to file prefix_line_orbits
void compute_spread_types_wrt_H(int verbose_level);
void compute_H_orbits_on_spreads(int verbose_level);
   // computes the orbits of H on spreads (NOT on lines!)
   // and writes to file fname_orbits
void test_orbits_on_spreads(int verbose_level);
void reduce_spreads(int verbose_level);
void compute_reduced_spread_types_wrt_H(int verbose_level);
   // Spread_types[P->nb_spreads * (group_order + 1)]
```

```
void compute_H_orbits_on_reduced_spreads(int verbose_level);
actions::action *restricted_action(
      int orbit_length, int verbose_level);
int test_if_pair_of_sets_of_reduced_spreads_are_adjacent(
   long int *orbit1, int len1,
   long int *orbit2, int len2,
   int verbose_level);
   // tests if every spread from set1
   // is line-disjoint from every spread from set2
   // using Spread_tables_reduced
void create_graph_and_save_to_file(
      std::string &fname,
   int orbit_length,
   int f_has_user_data,
   long int *user_data, int user_data_size,
   int verbose_level);
void create_graph_on_mixed_orbits_and_save_to_file(
      std::string &orbit_lengths_text,
      int f_has_user_data,
      long int *user_data, int user_data_size,
      int verbose_level);
int find_orbits_of_length_in_reduced_spread_table(
      int orbit_length);
void compute_orbit_invariant_on_classified_orbits(
      int verbose_level);
int evaluate_orbit_invariant_function(
      int a, int i, int j, int verbose_level);
void classify_orbit_invariant(int verbose_level);
void report_orbit_invariant(std::ostream &ost);
void report2(std::ostream &ost, int verbose_level);
void report(int verbose_level);
void report_line_orbits_under_H(
      std::ostream &ost, int verbose_level);
void get_spreads_in_reduced_orbits_by_type(int type_idx,
      int &nb_orbits, int &orbit_length,
      long int *&orbit_idx,
      long int *&spreads_in_reduced_orbits_by_type,
      int f_original_spread_numbers,
      int verbose_level);
void export_reduced_spread_orbits_csv(
      std::string &fname_base,
      int f_original_spread_numbers, int verbose_level);
void report_reduced_spread_orbits(
      std::ostream &ost,
      int f_original_spread_numbers, int verbose_level);
void report_good_spreads(
      std::ostream &ost);
```

```
};



// ##############################################################################
// packings_global.cpp
// ##############################################################################

//! classification and investigation of packings in PG(3,q)

class packings_global {
public:

   packings_global();
   ~packings_global();
   void merge_packings(
         std::string *fnames, int nb_files,
         std::string &file_of_spreads,
         data_structures::classify_bitvectors *&CB,
         int verbose_level);
   void select_packings(
         std::string &fname,
         std::string &file_of_spreads_original,
         geometry::spread_tables *Spread_tables,
         int f_self_polar,
         int f_ago, int select_ago,
         data_structures::classify_bitvectors *&CB,
         int verbose_level);
   void select_packings_self_dual(
         std::string &fname,
         std::string &file_of_spreads_original,
         int f_split, int split_r, int split_m,
         geometry::spread_tables *Spread_tables,
         data_structures::classify_bitvectors *&CB,
         int verbose_level);

};



// ##############################################################################
// regular_packing.cpp
// ##############################################################################

//! a regular packing as a partition of the Klein quadric into elliptic quadrics
```

```
class regular_packing {
public:
   packing_was *PW;

   std::vector<long int> External_lines;

   long int *spread_to_external_line_idx;
      // [T->nb_spreads]
      // spread_to_external_line_idx[i] is index into External_lines
      // corresponding to regular spread i
   long int *external_line_to_spread;
      // [nb_lines_orthogonal]
      // external_line_to_spread[i] is the index of the
      // regular spread of PG(3,q) in table T associated with
      // External_lines[i]


   regular_packing();
   ~regular_packing();
   void init(packing_was *PW, int verbose_level);

};




}}}




#endif /* SRC_LIB_TOP_LEVEL_PACKINGS_PACKINGS_H_ */
```

7.10 Projective Spaces

```
/*
 * projective_space.h
 *
 *  Created on: Mar 28, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_PROJECTIVE_SPACE_PROJECTIVE_SPACE_H_
#define SRC_LIB_TOP_LEVEL_PROJECTIVE_SPACE_PROJECTIVE_SPACE_H_



namespace orbiter {
namespace layer5_applications {
namespace projective_geometry {



// ##############################################################################
// canonical_form_classifier_description.cpp
// ##############################################################################



//! to classify objects using canonical forms


class canonical_form_classifier_description {

public:

   int f_space;
   std::string space_label;

   int f_input_fname_mask;
   std::string fname_mask;

   int f_nb_files;
   int nb_files;

   int f_output_fname;
   std::string fname_base_out;

   int f_label_po;
   std::string column_label_po;
   int f_label_so;
```

```
   std::string column_label_so;
   int f_label_equation;
   std::string column_label_eqn;
   int f_label_points;
   std::string column_label_pts;
   int f_label_lines;
   std::string column_label_bitangents;

   int f_degree;
   int degree;

   int f_algorithm_nauty;
   int f_algorithm_substructure;

   int f_substructure_size;
   int substructure_size;

   int f_skip;
   std::string skip_label;

   projective_space_with_action *PA;

   canonical_form_classifier *Canon_substructure;



   canonical_form_classifier_description();
   ~canonical_form_classifier_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// canonical_form_classifier.cpp
// ##############################################################################



//! to classify objects using canonical forms


class canonical_form_classifier {

public:
```

```
canonical_form_classifier_description *Descr;

int *skip_vector;
int skip_sz;

ring_theory::homogeneous_polynomial_domain *Poly_ring;

induced_actions::action_on_homogeneous_polynomials *AonHPD;

int nb_objects_to_test;

int idx_po, idx_so, idx_eqn, idx_pts, idx_bitangents;

// nauty stuff:

data_structures::classify_bitvectors *CB;
int canonical_labeling_len;

// substructure stuff:


// needed once for the whole classification process:
set_stabilizer::substructure_classifier *SubC;


// needed once for each object:
canonical_form_substructure **CFS_table;
   // [nb_objects_to_test]

canonical_form_of_variety **Variety_table;


int *Elt;
int *eqn2;


int counter;
int *Canonical_forms;
   // [nb_objects_to_test * Poly_ring->get_nb_monomials()]
long int *Goi; // [nb_objects_to_test]

data_structures::tally_vector_data
   *Classification_of_quartic_curves;
   // based on Canonical_forms, nb_objects_to_test

// transversal of the isomorphism types:
```

```
   int *transversal;
   int *frequency;
   int nb_types; // number of isomorphism types


   canonical_form_classifier();
   ~canonical_form_classifier();
   void init(
         canonical_form_classifier_description *Descr,
         int verbose_level);
   int skip_this_one(int counter);
   void count_nb_objects_to_test(int verbose_level);
   void classify(
         int verbose_level);
   void classify_nauty(int verbose_level);
   void classify_with_substructure(int verbose_level);
   void main_loop(int verbose_level);
   void prepare_input(int row,
         data_structures::spreadsheet *S,
         quartic_curve_object *&Qco, int verbose_level);
   void write_canonical_forms_csv(
         std::string &fname_base,
         int verbose_level);
   void generate_source_code(
         std::string &fname_base,
         int verbose_level);
   void report(
         poset_classification::poset_classification_report_options *Opt,
         int verbose_level);
   void report2(std::ostream &ost, int verbose_level);
   void export_canonical_form_data(
         std::string &fname, int verbose_level);

};


// ##############################################################################
// canonical_form_nauty.cpp
// ##############################################################################



//! to compute the canonical form of an object using nauty


class canonical_form_nauty {
```

```
public:

   canonical_form_classifier *Classifier;

   canonical_form_of_variety *Variety;
   //quartic_curve_object *Qco;

   int nb_rows, nb_cols;
   data_structures::bitvector *Canonical_form;
   long int *canonical_labeling;
   int canonical_labeling_len;


   groups::strong_generators *Set_stab;
      // the set stabilizer of the variety
      // this is not the stabilizer of the variety!

   orbits_schreier::orbit_of_equations *Orb;
      // orbit under the set stabilizer

   groups::strong_generators *Stab_gens_quartic;
      // the stabilizer of the original curve


   canonical_form_nauty();
   ~canonical_form_nauty();
   void init(
         canonical_form_classifier *Classifier,
         int verbose_level);
   void canonical_form_of_quartic_curve(
         canonical_form_of_variety *Variety,
         int verbose_level);
   // Computes the canonical labeling of the graph associated with
   // the set of rational points of the curve.
   // Computes the stabilizer of the set of rational points of the curve.
   // Computes the orbit of the equation under the stabilizer of the set.

};



// ##############################################################################
// canonical_form_of_variety.cpp
// ##############################################################################
```

```
//! to compute the canonical form of a variety

class canonical_form_of_variety {

public:

   canonical_form_classifier *Canonical_form_classifier;

   std::string fname_case_out;

   // input:
   quartic_curve_object *Qco;


   // output:
   long int *canonical_pts;
   int *canonical_equation;
   int *transporter_to_canonical_form;
   groups::strong_generators *gens_stab_of_canonical_equation;

   ring_theory::longinteger_object *go_eqn;


   canonical_form_of_variety();
   ~canonical_form_of_variety();
   void init(
         canonical_form_classifier *Canonical_form_classifier,
         std::string &fname_case_out,
         quartic_curve_object *Qco,
         int verbose_level);
   void classify_curve_nauty(
         int verbose_level);
   void handle_repeated_canonical_form_of_set(
         int idx,
         canonical_form_nauty *C,
         long int *alpha, int *gamma,
         int verbose_level);
   int find_equation(
         canonical_form_nauty *C,
         long int *alpha, int *gamma,
         int idx1, int &found_at,
         int verbose_level);
   void add_object_and_compute_canonical_equation(
         canonical_form_nauty *C,
         int idx, int verbose_level);
   void compute_canonical_form(
         int counter,
```

```
        int verbose_level);

};


// ##############################################################################
// canonical_form_substructure.cpp
// ##############################################################################



//! to compute the canonical form of an object using substructure canonization

class canonical_form_substructure {

public:

   canonical_form_of_variety *Variety;


   set_stabilizer::substructure_stats_and_selection *SubSt;

   set_stabilizer::compute_stabilizer *CS;

   groups::strong_generators *Gens_stabilizer_original_set;
   groups::strong_generators *Gens_stabilizer_canonical_form;


   orbits_schreier::orbit_of_equations *Orb;


   int *trans1;
   int *trans2;
   int *intermediate_equation;



   int *Elt;
   int *eqn2;



   canonical_form_substructure();
   ~canonical_form_substructure();
   void classify_curve_with_substructure(
         canonical_form_of_variety *Variety,
         int verbose_level);
```

```
   void handle_orbit(
         int *transporter_to_canonical_form,
         groups::strong_generators *&Gens_stabilizer_original_set,
         groups::strong_generators *&Gens_stabilizer_canonical_form,
         int verbose_level);


};


// ##############################################################################
// object_in_projective_space_with_action.cpp
// ##############################################################################



//! to represent an object in projective space


class object_in_projective_space_with_action {

public:

   geometry::object_with_canonical_form *OwCF;
      // do not free

   groups::strong_generators *Aut_gens;
      // generators for the automorphism group

   long int ago;
   int nb_rows, nb_cols;
   int *canonical_labeling;


   object_in_projective_space_with_action();
   ~object_in_projective_space_with_action();
   void init(
         geometry::object_with_canonical_form *OwCF,
         long int ago,
         groups::strong_generators *Aut_gens,
         int *canonical_labeling,
         int verbose_level);
   void print();
   void report(
         std::ostream &fp,
         projective_space_with_action *PA,
         int max_TDO_depth, int verbose_level);
```

```
};


// ##############################################################################
// projective_space_activity_description.cpp
// ##############################################################################

//! description of an activity associated with a projective space


class projective_space_activity_description {
public:


   int f_export_point_line_incidence_matrix;

   int f_table_of_cubic_surfaces_compute_properties;
   std::string table_of_cubic_surfaces_compute_fname_csv;
   int table_of_cubic_surfaces_compute_defining_q;
   int table_of_cubic_surfaces_compute_column_offset;

   int f_cubic_surface_properties_analyze;
   std::string cubic_surface_properties_fname_csv;
   int cubic_surface_properties_defining_q;

   int f_canonical_form_of_code;
   std::string canonical_form_of_code_label;
   std::string canonical_form_of_code_generator_matrix;
   combinatorics::classification_of_objects_description
      *Canonical_form_codes_Descr;

   int f_map;
   std::string map_ring_label;
   std::string map_formula_label;
   std::string map_parameters;

   int f_analyze_del_Pezzo_surface;
   std::string analyze_del_Pezzo_surface_label;
   std::string analyze_del_Pezzo_surface_parameters;

   int f_decomposition_by_element_PG;
   int decomposition_by_element_power;
   std::string decomposition_by_element_data;
   std::string decomposition_by_element_fname;
```

```
   int f_decomposition_by_subgroup;
   std::string decomposition_by_subgroup_label;
   groups::linear_group_description
      * decomposition_by_subgroup_Descr;

   int f_table_of_quartic_curves;
      // based on knowledge_base

   int f_table_of_cubic_surfaces;
      // based on knowledge_base


   int f_sweep;
   std::string sweep_fname;

   int f_sweep_4_15_lines;
   std::string sweep_4_15_lines_fname;
   applications_in_algebraic_geometry::cubic_surfaces_in_general::surface_create_d
escription
      *sweep_4_15_lines_surface_description;

   int f_sweep_F_beta_9_lines;
   std::string sweep_F_beta_9_lines_fname;
   applications_in_algebraic_geometry::cubic_surfaces_in_general::surface_create_d
escription
      *sweep_F_beta_9_lines_surface_description;

   int f_sweep_6_9_lines;
   std::string sweep_6_9_lines_fname;
   applications_in_algebraic_geometry::cubic_surfaces_in_general::surface_create_d
escription
      *sweep_6_9_lines_surface_description;

   int f_sweep_4_27;
   std::string sweep_4_27_fname;
   applications_in_algebraic_geometry::cubic_surfaces_in_general::surface_create_d
escription
      *sweep_4_27_surface_description;

   int f_sweep_4_L9_E4;
   std::string sweep_4_L9_E4_fname;
   applications_in_algebraic_geometry::cubic_surfaces_in_general::surface_create_d
escription
      *sweep_4_L9_E4_surface_description;

   int f_cheat_sheet;
```

```
int f_set_stabilizer;
int set_stabilizer_intermediate_set_size;
std::string set_stabilizer_fname_mask;
int set_stabilizer_nb;
std::string set_stabilizer_column_label;
std::string set_stabilizer_fname_out;

int f_conic_type;
int conic_type_threshold;
std::string conic_type_set_text;

int f_lift_skew_hexagon;
std::string lift_skew_hexagon_text;

int f_lift_skew_hexagon_with_polarity;
std::string lift_skew_hexagon_with_polarity_polarity;

int f_arc_with_given_set_as_s_lines_after_dualizing;
int arc_size;
int arc_d;
int arc_d_low;
int arc_s;
std::string arc_input_set;
std::string arc_label;

int f_arc_with_two_given_sets_of_lines_after_dualizing;
int arc_t;
std::string t_lines_string;

int f_arc_with_three_given_sets_of_lines_after_dualizing;
int arc_u;
std::string u_lines_string;

int f_dualize_hyperplanes_to_points;
int f_dualize_points_to_hyperplanes;
std::string dualize_input_set;

int f_dualize_rank_k_subspaces;
int dualize_rank_k_subspaces_k;

int f_lines_on_point_but_within_a_plane;
long int lines_on_point_but_within_a_plane_point_rk;
long int lines_on_point_but_within_a_plane_plane_rk;

int f_rank_lines_in_PG;
std::string rank_lines_in_PG_label;
```

```
int f_unrank_lines_in_PG;
std::string unrank_lines_in_PG_text;

int f_move_two_lines_in_hyperplane_stabilizer;
long int line1_from;
long int line2_from;
long int line1_to;
long int line2_to;

int f_move_two_lines_in_hyperplane_stabilizer_text;
std::string line1_from_text;
std::string line2_from_text;
std::string line1_to_text;
std::string line2_to_text;

int f_planes_through_line;
std::string planes_through_line_rank;

int f_restricted_incidence_matrix;
int restricted_incidence_matrix_type_row_objects;
int restricted_incidence_matrix_type_col_objects;
std::string restricted_incidence_matrix_row_objects;
std::string restricted_incidence_matrix_col_objects;
std::string restricted_incidence_matrix_file_name;

//int f_make_relation;
//long int make_relation_plane_rk;

int f_plane_intersection_type_of_klein_image;
int plane_intersection_type_of_klein_image_threshold;
std::string plane_intersection_type_of_klein_image_input;

int f_report_Grassmannian;
int report_Grassmannian_k;



// classification stuff:

int f_classify_surfaces_with_double_sixes;
std::string classify_surfaces_with_double_sixes_label;
std::string classify_surfaces_with_double_sixes_control_label;


int f_classify_surfaces_through_arcs_and_two_lines;

int f_test_nb_Eckardt_points;
```

```
   int nb_E;

   int f_classify_surfaces_through_arcs_and_trihedral_pairs;

   int f_trihedra1_control;
   poset_classification::poset_classification_control
      *Trihedra1_control;
   int f_trihedra2_control;
   poset_classification::poset_classification_control
      *Trihedra2_control;
   int f_control_six_arcs;
   std::string Control_six_arcs_label;

   int f_six_arcs_not_on_conic;
   int f_filter_by_nb_Eckardt_points;
   int nb_Eckardt_points;

   int f_classify_arcs;
   apps_geometry::arc_generator_description
      *Arc_generator_description;

   int f_classify_cubic_curves;



   int f_classify_semifields;
   semifields::semifield_classify_description
      *Semifield_classify_description;
   poset_classification::poset_classification_control
      *Semifield_classify_Control;

   int f_classify_bent_functions;
   int classify_bent_functions_n;


   projective_space_activity_description();
   ~projective_space_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();


};


// ##############################################################################
```

```
// projective_space_activity.cpp
// ##############################################################################

//! an activity associated with a projective space


class projective_space_activity {
public:

   projective_space_activity_description *Descr;

   projective_space_with_action *PA;


   projective_space_activity();
   ~projective_space_activity();
   void perform_activity(int verbose_level);
   void do_rank_lines_in_PG(
         std::string &label,
         int verbose_level);

};

// ##############################################################################
// projective_space_global.cpp
// ##############################################################################

//! collection of worker functions for projective space


class projective_space_global {
public:
   void analyze_del_Pezzo_surface(
         projective_space_with_action *PA,
         std::string &label,
         std::string &evaluate_text,
         int verbose_level);
   void analyze_del_Pezzo_surface_formula_given(
         projective_space_with_action *PA,
         expression_parser::formula *F,
         std::string &evaluate_text,
         int verbose_level);
   void do_lift_skew_hexagon(
         projective_space_with_action *PA,
         std::string &text,
         int verbose_level);
   void do_lift_skew_hexagon_with_polarity(
```

```
         projective_space_with_action *PA,
         std::string &polarity_36,
         int verbose_level);
   void do_classify_arcs(
         projective_space_with_action *PA,
         apps_geometry::arc_generator_description
            *Arc_generator_description,
         int verbose_level);
   void do_classify_cubic_curves(
         projective_space_with_action *PA,
         apps_geometry::arc_generator_description
            *Arc_generator_description,
         int verbose_level);
   void set_stabilizer(
         projective_space_with_action *PA,
         int intermediate_subset_size,
         std::string &fname_mask, int nb, std::string &column_label,
         std::string &fname_out,
         int verbose_level);
#if 0
   void make_relation(
         projective_space_with_action *PA,
         long int plane_rk,
         int verbose_level);
#endif
   void classify_bent_functions(
         projective_space_with_action *PA,
         int n,
         int verbose_level);

};




// ##############################################################################
// projective_space_with_action_description.cpp
// ##############################################################################


//! description of a projective space with action

class projective_space_with_action_description {
public:

   int f_n;
   int n;
```

```
   int f_q;
   int q;

   int f_field;
   std::string field_label;

   field_theory::finite_field *F;

   int f_use_projectivity_subgroup;

   int f_override_verbose_level;
   int override_verbose_level;

   projective_space_with_action_description();
   ~projective_space_with_action_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// projective_space_with_action.cpp
// ##############################################################################




//! projective space PG(n,q) with automorphism group PGGL(n+1,q)



class projective_space_with_action {

public:

   int n; // projective dimension
   int d; // n + 1
   int q;
   field_theory::finite_field *F; // do not free
   int f_semilinear;
   int f_init_incidence_structure;
```

```
   geometry::projective_space *P;

   // if n >= 3:
   projective_space_with_action *PA2;

   // if n == 3
   applications_in_algebraic_geometry::cubic_surfaces_in_general::surface_with_act
ion
      *Surf_A;


   // if n == 2:
   algebraic_geometry::quartic_curve_domain *Dom;
   applications_in_algebraic_geometry::quartic_curves::quartic_curve_domain_with_a
ction
      *QCDA;


   actions::action *A; // linear group PGGL(d,q) in the action on points
   actions::action *A_on_lines; // linear group PGGL(d,q) acting on lines

   int f_has_action_on_planes;
   actions::action *A_on_planes; // linear group PGGL(d,q) acting on planes


   int *Elt1;


   projective_space_with_action();
   ~projective_space_with_action();
   void init(
         field_theory::finite_field *F,
         int n, int f_semilinear,
      int f_init_incidence_structure, int verbose_level);
   void init_group(int f_semilinear, int verbose_level);
   void canonical_labeling(
         geometry::object_with_canonical_form *OiP,
      int *canonical_labeling,
      int verbose_level);
   void report_fixed_points_lines_and_planes(
      int *Elt, std::ostream &ost,
      int verbose_level);
   void report_orbits_on_points_lines_and_planes(
      int *Elt, std::ostream &ost,
      int verbose_level);
   void compute_group_of_set(long int *set, int set_sz,
         groups::strong_generators *&Sg,
```

```
        int verbose_level);
      // ToDo
   void do_cheat_sheet_for_decomposition_by_element_PG(
        int decomposition_by_element_power,
        std::string &decomposition_by_element_data,
        std::string &fname_base,
        int verbose_level);
   void do_cheat_sheet_for_decomposition_by_subgroup(
        std::string &label,
        groups::linear_group_description * subgroup_Descr,
        int verbose_level);
   void report(
      std::ostream &ost,
      graphics::layered_graph_draw_options *O,
      int verbose_level);
   void canonical_form_of_code(
        std::string &label,
        int *genma, int m, int n,
        combinatorics::classification_of_objects_description
           *Canonical_form_codes_Descr,
        int verbose_level);
   void table_of_quartic_curves(int verbose_level);
   void table_of_cubic_surfaces(int verbose_level);
   void cheat_sheet(
        graphics::layered_graph_draw_options *O,
        int verbose_level);
   void do_spread_classify(int k,
        poset_classification::poset_classification_control
           *Control,
        int verbose_level);
   void setup_surface_with_action(
        applications_in_algebraic_geometry::cubic_surfaces_in_general::surface_with
_action
           *&Surf_A,
        int verbose_level);
   void report_decomposition_by_group(
        groups::strong_generators *SG,
        std::ostream &ost, std::string &fname_base,
      int verbose_level);


};

// ##############################################################################
// quartic_curve_object.cpp
// ##############################################################################
```

```
//! a quartic curve object with bitangents and equation


class quartic_curve_object {

public:

   int cnt;
   int po;
   int so;

   int *eqn;
   int sz;

   long int *pts;
   int nb_pts;

   long int *bitangents;
   int nb_bitangents;


   quartic_curve_object();
   ~quartic_curve_object();
   void init(
         int cnt, int po, int so,
         std::string &eqn_txt,
         std::string &pts_txt, std::string &bitangents_txt,
         int verbose_level);
   void init_image_of(quartic_curve_object *old_one,
         int *Elt,
         actions::action *A,
         actions::action *A_on_lines,
         int *eqn2,
         int verbose_level);
   void print(std::ostream &ost);

};



}}}
```

```
#endif /* SRC_LIB_TOP_LEVEL_PROJECTIVE_SPACE_PROJECTIVE_SPACE_H_ */
```

7.11 Semifields

```
/*
 * semifields.h
 *
 *  Created on: Nov 4, 2020
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_SEMIFIELDS_SEMIFIELDS_H_
#define SRC_LIB_TOP_LEVEL_SEMIFIELDS_SEMIFIELDS_H_



namespace orbiter {
namespace layer5_applications {
namespace semifields {

// ##############################################################################
// semifield_classify_description.cpp
// ##############################################################################


//! description of a semifield classification problem

class semifield_classify_description {
public:

   int f_order;
   int order;
   int f_dim_over_kernel;
   int dim_over_kernel;
   int f_prefix;
   std::string prefix;
   int f_orbits_light;
   int f_test_semifield;
   std::string test_semifield_data;
   int f_identify_semifield;
   std::string identify_semifield_data;
   int f_identify_semifields_from_file;
   std::string identify_semifields_from_file_fname;
   int f_load_classification;
   int f_report;
   int f_decomposition_matrix_level_3;

   int f_level_two_prefix;
   std::string level_two_prefix;
   int f_level_three_prefix;
```

```
   std::string level_three_prefix;


   semifield_classify_description();
   ~semifield_classify_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);

};



// ##############################################################################
// semifield_classify_with_substructure.cpp
// ##############################################################################


//! classification of semifields using substructure

class semifield_classify_with_substructure {
public:

   int t0;

   semifield_classify_description *Descr;

   projective_geometry::projective_space_with_action *PA;
   groups::matrix_group *Mtx;
   poset_classification::poset_classification_control *Control;


   int *identify_semifields_from_file_Po;
   int identify_semifields_from_file_m;


   int f_trace_record_prefix;
   std::string trace_record_prefix;
   int f_FstLen;
   std::string fname_FstLen;
   int f_Data;
   std::string fname_Data;

   int p, e, e1, n, k, q, k2;

   semifield_substructure *Sub;
   semifield_level_two *L2;
```

```
   int nb_existing_cases;
   int *Existing_cases;
   int *Existing_cases_fst;
   int *Existing_cases_len;


   int nb_non_unique_cases;
   int *Non_unique_cases;
   int *Non_unique_cases_fst;
   int *Non_unique_cases_len;
   long int *Non_unique_cases_go;


   invariant_relations::classification_step *Semifields;


   semifield_classify_with_substructure();
   ~semifield_classify_with_substructure();
   void init(
         semifield_classify_description *Descr,
         projective_geometry::projective_space_with_action *PA,
         poset_classification::poset_classification_control *Control,
         int verbose_level);
   void read_data(int verbose_level);
   void create_fname_for_classification(char *fname);
   void create_fname_for_flag_orbits(char *fname);
   void classify_semifields(int verbose_level);
   void load_classification(int verbose_level);
   void load_flag_orbits(int verbose_level);
   void identify_semifield(int verbose_level);
   void identify_semifields_from_file(int verbose_level);
   void latex_report(int verbose_level);
   void generate_source_code(int verbose_level);
   void decomposition(int verbose_level);
};




// ##############################################################################
// semifield_classify.cpp
// ##############################################################################
```

```
//! classification of semifields using poset classification

class semifield_classify {
public:

   projective_geometry::projective_space_with_action *PA;

   int n;
   int k;
   int k2; // = k * k
   //linear_group *LG;
   groups::matrix_group *Mtx;

   int q;
   int order; // q^k


   std::string level_two_prefix;

   std::string level_three_prefix;


   geometry::spread_domain *SD;
   spreads::spread_classify *T;

   actions::action *A; // = T->A = PGL_n_q
   int *Elt1;
   groups::sims *G; // = T->R->A0_linear->Sims

   actions::action *A0;
   actions::action *A0_linear;


   induced_actions::action_on_spread_set *A_on_S;
   actions::action *AS;

   groups::strong_generators *Strong_gens;
      // the stabilizer of two components in a spread:
      // infinity and zero


   poset_classification::poset_with_group_action *Poset;
   poset_classification::poset_classification_control *Control;

   poset_classification::poset_classification *Gen;
   groups::sims *Symmetry_group;
```

```
int vector_space_dimension; // = k * k
int schreier_depth;

// for test_partial_semifield:
int *test_base_cols; // [n]
int *test_v; // [n]
int *test_w; // [k2]
int *test_Basis; // [k * k2]

// for knuth operations:
int *Basis1; // [k * k2]
int *Basis2; // [k * k2]

// for compute_orbit_of_subspaces:
int *desired_pivots;

semifield_classify();
~semifield_classify();
void init(
      projective_geometry::projective_space_with_action *PA,
      int k,
      poset_classification::poset_classification_control *Control,
      std::string &level_two_prefix,
      std::string &level_three_prefix,
      int verbose_level);
void report(std::ostream &ost, int level,
      semifield_level_two *L2,
      semifield_lifting *L3,
      graphics::layered_graph_draw_options *draw_options,
      int verbose_level);
void init_poset_classification(
      poset_classification::poset_classification_control *Control,
      int verbose_level);
void compute_orbits(int depth, int verbose_level);
void list_points();
long int rank_point(int *v, int verbose_level);
void unrank_point(int *v, long int rk, int verbose_level);
void early_test_func(long int *S, int len,
   long int *candidates, int nb_candidates,
   long int *good_candidates, int &nb_good_candidates,
   int verbose_level);
int test_candidate(
      int **Mtx_stack, int stack_size, int *M,
      int verbose_level);
int test_partial_semifield_numerical_data(
      long int *data, int data_sz, int verbose_level);
```

```
   int test_partial_semifield(
         int *Basis, int n, int verbose_level);
   void test_rank_unrank();
   void matrix_unrank(long int rk, int *Mtx);
   long int matrix_rank(int *Mtx);
   long int matrix_rank_without_first_column(int *Mtx);
   void basis_print(int *Mtx, int sz);
   void basis_print_numeric(long int *Rk, int sz);
   void matrix_print(int *Mtx);
   void matrix_print_numeric(long int rk);
   void print_set_of_matrices_numeric(long int *Rk, int nb);
   void apply_element(int *Elt,
      int *basis_in, int *basis_out,
      int first, int last_plus_one, int verbose_level);
   void apply_element_and_copy_back(int *Elt,
      int *basis_in, int *basis_out,
      int first, int last_plus_one, int verbose_level);
   int test_if_third_basis_vector_is_ok(int *Basis);
   void candidates_classify_by_first_column(
      long int *Input_set, int input_set_sz,
      int window_bottom, int window_size,
      long int **&Set, int *&Set_sz, int &Nb_sets,
      int verbose_level);
   void make_fname_candidates_at_level_two_orbit(
         std::string &fname, int orbit);
   void make_fname_candidates_at_level_two_orbit_txt(
         std::string &fname, int orbit);
   void make_fname_candidates_at_level_three_orbit(
         std::string &fname, int orbit);
   void make_fname_candidates_at_level_two_orbit_by_type(
         std::string &fname, int orbit, int h);
   void compute_orbit_of_subspaces(
      long int *input_data,
      groups::strong_generators *stabilizer_gens,
      orbits_schreier::orbit_of_subspaces *&Orb,
      int verbose_level);
   // allocates an orbit_of_subspaces data structure in Orb
   void init_desired_pivots(int verbose_level);
   void knuth_operation(int t,
         long int *data_in, long int *data_out,
         int verbose_level);
};


// ##############################################################################
// semifield_level_two.cpp
```

```
// ##############################################################################


//! The first and second steps in classifying semifields


class semifield_level_two {
public:
   semifield_classify *SC;
   int n; // = 2 * k
   int k;
   int k2;
   int q;

   actions::action *A; // PGL(n,q)
   actions::action *A_PGLk; // PGL(k,q)
   groups::matrix_group *M;
   field_theory::finite_field *F;
   algebra::gl_classes *C;
   int *desired_pivots; // [k]


   // Level one:
   algebra::gl_class_rep *R; // [nb_classes]
      // conjugacy class reps,
      // allocated and computed in C->make_classes,
      // which is called from downstep()
   int nb_classes;

   int *Basis, *Mtx, *Mtx_Id, *Mtx_2, *Elt, *Elt2;

   // the following arrays are all [nb_classes]
   long int *class_rep_rank;
   long int *class_rep_plus_I_rank;
   int **class_rep_plus_I_Basis;
      // computed via C->identify_matrix
      // aplied to the matrix representing the conjugacy class
      // plus the identity matrix
      // if the two matrices A and A + I belong to the same conjugacy class,
      // then the matrix class_rep_plus_I_Basis will be added to the
      // centralizer to form the stabilizer of the flag.
   int **class_rep_plus_I_Basis_inv;
   int *R_i_plus_I_class_idx;
   int *class_to_flag_orbit;
      // class_to_flag_orbit[i] is the flag orbit which contains class i
```

```
int nb_flag_orbits; // the number of flag orbits
groups::strong_generators *Flag_orbit_stabilizer; // [nb_flag_orbits]
int *flag_orbit_classes;  // [nb_flag_orbits * 2]
   // for each flag orbit i,
   // the conjugacy class associated to R_i and R_i + I, respectively
int *flag_orbit_number_of_matrices; // [nb_flag_orbits]
int *flag_orbit_length; // [nb_flag_orbits]
int *f_Fusion; // [nb_flag_orbits]
int *Fusion_idx; // [nb_flag_orbits]
int **Fusion_elt; // [nb_flag_orbits]

int nb_orbits;
int *defining_flag_orbit; // same as Fo
   // The flag orbit which led to the definition
   // of this orbit representative.
   // To get the actual rep a, do
   // idx = flag_orbit_classes[ext * 2 + 0];
   // a = class_rep_rank[idx];
   // where ext = up_orbit_rep[i]

   //int *Po; // [nb_orbits]
   // There is only one orbit at level one,
   // so there is no need to store Po

// it does not have a Po[]

int *So;
   // [nb_orbits] So[i] is the index of the conjugacy class
   // associated with the flag orbit Fo[i]
   // So[i] = flag_orbit_classes[Fo[i] * 2 + 0];

int *Fo;
   // [nb_orbits]
   // Fo[i] is the index of the flag orbit
   // which let to the definition of orbit i

long int *Go; // [nb_orbits]
long int *Pt; // [nb_orbits]


//for (i = 0; i < nb_orbits; i++) {
//ext = defining_flag_orbit[i];
//idx = flag_orbit_classes[ext * 2 + 0];
//a = class_rep_rank[idx];
//b = class_rep_plus_I_rank[idx];
//Fo[i] = ext;
//So[i] = idx;
```

```
//Pt[i] = a;
//Go[i] = go.as_lint();


groups::strong_generators *Stabilizer_gens;
   // stabilizer generators for the
   // chosen orbit representatives at level two

int *E1, *E2, *E3, *E4;
//int *Mnn;
int *Mtx1, *Mtx2, *Mtx3, *Mtx4, *Mtx5, *Mtx6;
int *ELT1, *ELT2, *ELT3;
int *M1;
int *Basis1, *Basis2;

algebra::gl_class_rep *R1, *R2;

long int **Candidates;
   // candidates for the generator matrix,
   // [nb_orbits]
int *Nb_candidates;
   // [nb_orbits]


semifield_level_two();
~semifield_level_two();
void init(semifield_classify *SC, int verbose_level);
void init_desired_pivots(int verbose_level);
void list_all_elements_in_conjugacy_class(
      int c, int verbose_level);
void compute_level_two(int nb_stages, int verbose_level);
void downstep(int verbose_level);
void compute_stabilizers_downstep(int verbose_level);
void upstep(int verbose_level);
void trace(int f, int coset,
      long int a, long int b, int &f_automorphism, int *&Aut,
      int verbose_level);
void multiply_to_the_right(
      int *ELT1, int *Mtx, int *ELT2, int *ELT3,
      int verbose_level);
   // Creates the n x n matrix which is the 2 x 2 block matrix
   // (A 0)
   // (0 A)
   // where A is Mtx.
   // The resulting element is stored in ELT2.
   // After this, ELT1 * ELT2 will be stored in ELT3
void compute_candidates_at_level_two_case(
```

```
      int orbit,
      long int *&Candidates, int &nb_candidates,
      int verbose_level);
   void allocate_candidates_at_level_two(
         int verbose_level);
   int test_if_file_exists_candidates_at_level_two_case(
      int orbit, int verbose_level);
   int test_if_txt_file_exists_candidates_at_level_two_case(
      int orbit, int verbose_level);
   void find_all_candidates_at_level_two(
         int verbose_level);
   void read_candidates_at_level_two_case(
      long int *&Candidates, int &Nb_candidates, int orbit,
      int verbose_level);
   void read_candidates_at_level_two_case_txt_file(
      long int *&Candidates, int &Nb_candidates, int orbit,
      int verbose_level);
   void write_candidates_at_level_two_case(
      long int *Candidates, int Nb_candidates, int orbit,
      int verbose_level);
   void read_candidates_at_level_two_by_type(
         data_structures::set_of_sets_lint *&Candidates_by_type,
         int orbit,
         int verbose_level);
   void get_basis_and_pivots(int po,
         int *basis, int *pivots, int verbose_level);
   void report(std::ofstream &ost, int verbose_level);
   void create_fname_level_info_file(std::string &fname);
   void write_level_info_file(int verbose_level);
   void read_level_info_file(int verbose_level);
};


// ##############################################################################
// semifield_lifting.cpp
// ##############################################################################


//! One step of lifting for classifying semifields


class semifield_lifting {
public:
   semifield_classify *SC;
   semifield_level_two *L2; // only if cur_level == 3
   semifield_lifting *Prev; // only if cur_level > 3
   int n;
```

```
int k;
int k2;

int cur_level;
int prev_level_nb_orbits;

int f_prefix;
std::string prefix;

groups::strong_generators *Prev_stabilizer_gens;
long int **Candidates;
   // candidates for the generator matrix,
   // [nb_orbits]
int *Nb_candidates;
   // [nb_orbits]


semifield_downstep_node *Downstep_nodes;

int nb_flag_orbits;

int *flag_orbit_first; // [prev_level_nb_orbits]
int *flag_orbit_len; // [prev_level_nb_orbits]


semifield_flag_orbit_node *Flag_orbits;

geometry::grassmann *Gr;

// po = primary orbit
// so = secondary orbit
// mo = middle orbit = flag orbit
// pt = point
int nb_orbits;
int *Po; // [nb_orbits]
int *So; // [nb_orbits]
int *Mo; // [nb_orbits]
long int *Go; // [nb_orbits]
long int *Pt; // [nb_orbits]
groups::strong_generators *Stabilizer_gens; // [nb_orbits]

// deep_search:
int *Matrix0, *Matrix1, *Matrix2;
int *window_in;

// for trace_very_general:
int *ELT1, *ELT2, *ELT3;
```

```
int *basis_tmp;
int *base_cols;
int *M1;
int *Basis;
algebra::gl_class_rep *R1;



semifield_lifting();
~semifield_lifting();
void init_level_three(semifield_level_two *L2,
      int f_prefix, std::string &prefix,
      int verbose_level);
void report(std::ostream &ost, int verbose_level);
void recover_level_three_downstep(int verbose_level);
void recover_level_three_from_file(
      int f_read_flag_orbits, int verbose_level);
void compute_level_three(int verbose_level);
void level_two_down(int verbose_level);
void level_two_flag_orbits(int verbose_level);
void level_two_upstep(int verbose_level);
void downstep(
   int level,
   int verbose_level);
// level is the previous level
void compute_flag_orbits(
   int level,
   int verbose_level);
// level is the previous level
void upstep(
   int level,
   int verbose_level);
// level is the level that we want to classify
void upstep_loop_over_down_set(
   int level, int f, int po, int so, int N,
   int *transporter, int *Mtx, //int *pivots,
   int *base_change_matrix,
   int *changed_space,
   //int *changed_space_after_trace,
   long int *set,
   int **Aut,
   int verbose_level);
// level is the level that we want to classify
void find_all_candidates(
   int level,
   int verbose_level);
void get_basis(
```

```
   int po3, int *basis,
   int verbose_level);
groups::strong_generators *get_stabilizer_generators(
   int level, int orbit_idx,
   int verbose_level);
int trace_to_level_three(
   int *input_basis, int basis_sz, int *transporter,
   int &trace_po,
   int verbose_level);
int trace_step_up(
   int &po, int &so,
   int *changed_basis, int basis_sz, int *basis_tmp,
   int *transporter, int *ELT3,
   int verbose_level);
void trace_very_general(
   int *input_basis, int basis_sz,
   int *transporter,
   int &trace_po, int &trace_so,
   int verbose_level);
   // input basis is input_basis of size basis_sz x k2
   // there is a check if input_basis defines a semifield
void trace_to_level_two(
   int *input_basis, int basis_sz,
   int *transporter,
   int &trace_po,
   int verbose_level);
// input basis is input_basis of size basis_sz x k2
// there is a check if input_basis defines a semifield
void deep_search(
   int orbit_r, int orbit_m,
   int f_out_path, std::string &out_path,
   int verbose_level);
void deep_search_at_level_three(
   int orbit_r, int orbit_m,
   int f_out_path, std::string &out_path,
   int &nb_sol,
   int verbose_level);
void print_stabilizer_orders();
void deep_search_at_level_three_orbit(
   int orbit, int *Basis, int *pivots,
   std::ofstream &fp,
   int &nb_sol,
   int verbose_level);
int candidate_testing(
   int orbit,
   int *last_mtx, int window_bottom, int window_size,
   data_structures::set_of_sets_lint *C_in,
```

```
      data_structures::set_of_sets_lint *C_out,
      long int *Tmp1, long int *Tmp2,
      int verbose_level);
   void level_three_get_a1_a2_a3(
      int po3, long int &a1, long int &a2, long int &a3,
      int verbose_level);
   void write_level_info_file(int verbose_level);
   void read_level_info_file(int verbose_level);
   void make_fname_flag_orbits(std::string &fname);
   void save_flag_orbits(int verbose_level);
   void read_flag_orbits(int verbose_level);
   void save_stabilizers(int verbose_level);
   void read_stabilizers(int verbose_level);
   void make_file_name_schreier(std::string &fname,
         int level, int orbit_idx);
   void create_fname_level_info_file(std::string &fname);
   void make_fname_stabilizers(std::string &fname);
   void make_fname_deep_search_slice_solutions(
         std::string &fname,
         int f_out_path, std::string &out_path,
         int orbit_r, int orbit_m);
   void make_fname_deep_search_slice_success(
         std::string &fname,
         int f_out_path, std::string &out_path,
         int orbit_r, int orbit_m);

};


// ##############################################################################
// semifield_substructure.cpp
// ##############################################################################

//! auxiliary class for classifying semifields using a three-dimensional substruc
ture


class semifield_substructure {
public:
   semifield_classify_with_substructure *SCWS;
   semifield_classify *SC;
   semifield_lifting *L3;
   geometry::grassmann *Gr3;
   geometry::grassmann *Gr2;
   int *Non_unique_cases_with_non_trivial_group;
   int nb_non_unique_cases_with_non_trivial_group;
```

```
int *Need_orbits_fst;
int *Need_orbits_len;

trace_record *TR;
int N; // = number of 3-dimensional subspaces
int N2; // = number of 2-dimensional subspaces
int f;
long int *Data;
int nb_solutions;
int data_size;
int start_column;
int *FstLen;
int *Len;
int nb_orbits_at_level_3;
int nb_orb_total; // = sum_i Nb_orb[i]
orbits_schreier::orbit_of_subspaces ***All_Orbits;
   // [nb_non_unique_cases_with_non_trivial_group]
int *Nb_orb; // [nb_non_unique_cases_with_non_trivial_group]
   // Nb_orb[i] is the number of orbits in All_Orbits[i]
int **Orbit_idx; // [nb_non_unique_cases_with_non_trivial_group]
   // Orbit_idx[i][j] = b
   // means that the j-th solution of Nontrivial case i
   // belongs to orbt All_Orbits[i][b]
int **Position; // [nb_non_unique_cases_with_non_trivial_group]
   // Position[i][j] = a
   // means that the j-th solution of Nontrivial case i
   // is the a-th element in All_Orbits[i][b]
   // where Orbit_idx[i][j] = b
int *Fo_first; // [nb_orbits_at_level_3]
int nb_flag_orbits;
invariant_relations::flag_orbits *Flag_orbits;
   // [nb_flag_orbits]
long int *data1;
long int *data2;
int *Basis1;
int *Basis2;
//int *Basis3;
int *B;
int *v1;
int *v2;
int *v3;
int *transporter1;
int *transporter2;
int *transporter3;
int *Elt1;
data_structures_groups::vector_ge *coset_reps;
```

```
   semifield_substructure();
   ~semifield_substructure();
   void init();
   void compute_cases(
         int nb_non_unique_cases,
         int *Non_unique_cases, long int *Non_unique_cases_go,
         int verbose_level);
   void compute_orbits(int verbose_level);
   void compute_flag_orbits(int verbose_level);
   void do_classify(int verbose_level);
   void loop_over_all_subspaces(
         int *f_processed, int &nb_processed,
         int verbose_level);
   void all_two_dimensional_subspaces(
         int *Trace_po, int verbose_level);
      // Trace_po[N2]
   int find_semifield_in_table(
      int po,
      long int *given_data,
      int &idx,
      int verbose_level);
   int identify(long int *data,
         int &rk, int &trace_po, int &fo, int &po,
         int *transporter,
         int verbose_level);
};




// ##############################################################################
// semifield_downstep_node.cpp
// ##############################################################################

//! auxiliary class for classifying semifields


class semifield_downstep_node {
public:
   semifield_classify *SC;
   semifield_lifting *SL;
   field_theory::finite_field *F;
   int k;
   int k2;

   int level;
   int orbit_number;
```

```
   long int *Candidates;
   int nb_candidates;

   int *subspace_basis;

   induced_actions::action_on_cosets *on_cosets;
   actions::action *A_on_cosets;

   groups::schreier *Sch;

   int first_flag_orbit;

   semifield_downstep_node();
   ~semifield_downstep_node();
   void init(
         semifield_lifting *SL,
         int level, int orbit_number,
      long int *Candidates, int nb_candidates, int first_flag_orbit,
      int verbose_level);
   int find_point(long int a);

};



// ##############################################################################
// semifield_flag_orbit_node.cpp
// ##############################################################################


//! auxiliary class for classifying semifields

class semifield_flag_orbit_node {
public:
   int downstep_primary_orbit;
   int downstep_secondary_orbit;
   int pt_local;
   long int pt;
   int downstep_orbit_len;
   int f_long_orbit;
   int upstep_orbit; // if !f_fusion_node
   int f_fusion_node;
   int fusion_with;
   int *fusion_elt;

   ring_theory::longinteger_object go;
```

```
   groups::strong_generators *gens;

   semifield_flag_orbit_node();
   ~semifield_flag_orbit_node();
   void init(
         int downstep_primary_orbit, int downstep_secondary_orbit,
      int pt_local, long int pt, int downstep_orbit_len, int f_long_orbit,
      int verbose_level);
   void group_order(ring_theory::longinteger_object &go);
   int group_order_as_int();
   void write_to_file_binary(
         semifield_lifting *SL, std::ofstream &fp,
         int verbose_level);
   void read_from_file_binary(
         semifield_lifting *SL, std::ifstream &fp,
         int verbose_level);

};

// ##############################################################################
// semifield_trace.cpp
// ##############################################################################


//! auxiliary class for isomorph recognition of a semifield

class semifield_trace {
public:
   semifield_classify *SC;
   semifield_lifting *SL;
   semifield_level_two *L2;
   actions::action *A;
   field_theory::finite_field *F;
   int n;
   int k;
   int k2;
   int *ELT1, *ELT2, *ELT3;
   int *M1;
   int *Basis;
   int *basis_tmp;
   int *base_cols;
   algebra::gl_class_rep *R1;

   semifield_trace();
   ~semifield_trace();
   void init(semifield_lifting *SL);
   void trace_very_general(
```

```
      int cur_level,
      int *input_basis, int basis_sz,
      int *basis_after_trace, int *transporter,
      int &trace_po, int &trace_so,
      int verbose_level);
      // input basis is input_basis of size basis_sz x k2
      // there is a check if input_basis defines a semifield
};


// ##############################################################################
// trace_record.cpp
// ##############################################################################


//! to record the result of isomorphism testing


class trace_record {
public:
   int coset;
   int trace_po;
   int f_skip;
   int solution_idx;
   int nb_sol;
   long int go;
   int pos;
   int so;
   int orbit_len;
   int f2;
   trace_record();
   ~trace_record();
};

void save_trace_record(
      trace_record *T,
      int f_trace_record_prefix, std::string &trace_record_prefix,
      int iso, int f, int po, int so, int N);


}}}

#endif /* SRC_LIB_TOP_LEVEL_SEMIFIELDS_SEMIFIELDS_H_ */
```

7.12 Spreads

```
/*
 * spreads.h
 *
 *  Created on: May 25, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_SPREADS_SPREADS_H_
#define SRC_LIB_TOP_LEVEL_SPREADS_SPREADS_H_


namespace orbiter {
namespace layer5_applications {
namespace spreads {


// ##############################################################################
// recoordinatize.cpp
// ##############################################################################

//! three skew lines in PG(3,q), used to classify spreads


class recoordinatize {
public:
   geometry::spread_domain *SD;

   int n;
   int k;
   int q;
   geometry::grassmann *Grass;
   field_theory::finite_field *F;

   actions::action *A;
      // P Gamma L(n,q)
   actions::action *A2;
      // action of A on grassmannian
      // of k-subspaces of V(n,q)
   int f_projective;
   int f_semilinear;
   int nCkq; // n choose k in q
   int (*check_function_incremental)(int len, long int *S,
      void *check_function_incremental_data, int verbose_level);
   void *check_function_incremental_data;

   //std::string fname_live_points;
```

```
int f_data_is_allocated;
int *M; // [(3 * k) * n]
int *M1; // [(3 * k) * n]
int *AA; // [n * n]
int *AAv; // [n * n]
int *TT; // [k * k]
int *TTv; // [k * k]
int *B; // [n * n]
int *C; // [n * n + 1]
int *N; // [(3 * k) * n]
int *Elt; // [A->elt_size_in_int]

// initialized in compute_starter():
long int starter_j1, starter_j2, starter_j3;
actions::action *A0;
   // P Gamma L(k,q)
actions::action *A0_linear;
   // PGL(k,q), needed for compute_live_points
data_structures_groups::vector_ge *gens2;

long int *live_points;
int nb_live_points;


recoordinatize();
~recoordinatize();
void init(
      geometry::spread_domain *SD,
      actions::action *A, actions::action *A2,
   int f_projective, int f_semilinear,
   int (*check_function_incremental)(
         int len, long int *S,
      void *data, int verbose_level),
   void *check_function_incremental_data,
   int verbose_level);
void do_recoordinatize(
      long int i1, long int i2, long int i3,
      int verbose_level);
void compute_starter(
      long int *&S, int &size,
      groups::strong_generators *&Strong_gens,
      int verbose_level);
void stabilizer_of_first_three(
      groups::strong_generators *&Strong_gens,
   int verbose_level);
```

```
   void compute_live_points(int verbose_level);
   void compute_live_points_low_level(
         long int *&live_points,
      int &nb_live_points, int verbose_level);
   int apply_test(
         long int *set, int sz, int verbose_level);
   void make_first_three(
         long int &j1, long int &j2, long int &j3,
         int verbose_level);
};

// ##############################################################################
// spread_activity_description.cpp
// ##############################################################################

//! description of an activity regarding a spread



class spread_activity_description {

public:

   int f_report;

   spread_activity_description();
   ~spread_activity_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();


};



// ##############################################################################
// spread_activity.cpp
// ##############################################################################

//! an activity regarding a spread



class spread_activity {

public:
```

```
   spread_activity_description *Descr;
   spread_create *Spread_create;
   geometry::spread_domain *SD;

   actions::action *A;
      // P Gamma L(n,q)
   actions::action *A2;
      // action of A on grassmannian
      // of k-subspaces of V(n,q)
   induced_actions::action_on_grassmannian *AG;

   actions::action *AGr;


   spread_activity();
   ~spread_activity();
   void init(
         spread_activity_description *Descr,
         spread_create *Spread_create,
         int verbose_level);
   void perform_activity(int verbose_level);
   void report(int verbose_level);
   void report2(std::ostream &ost, int verbose_level);

};







// ##############################################################################
// spread_classify_activity_description.cpp
// ##############################################################################

//! description of an activity regarding the classification of spreads



class spread_classify_activity_description {

public:


   int f_compute_starter;
```

```
   int f_prepare_lifting_single_case;
   int prepare_lifting_single_case_case_number;

   int f_prepare_lifting_all_cases;

   int f_split;
   int split_r;
   int split_m;

   int f_isomorph;
   layer4_classification::isomorph::isomorph_arguments
      *Isomorph_arguments;

   spread_classify_activity_description();
   ~spread_classify_activity_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};



// ##############################################################################
// spread_classify_activity.cpp
// ##############################################################################

//! an activity regarding the classification of spreads



class spread_classify_activity {

public:

   spread_classify_activity_description *Descr;
   spread_classify *Spread_classify;

   spread_classify_activity();
   ~spread_classify_activity();
   void init(
         spread_classify_activity_description *Descr,
         spread_classify *Spread_classify,
         int verbose_level);
   void perform_activity(int verbose_level);
```

```
};



// ##############################################################################
// spread_classify_description.cpp
// ##############################################################################



//! parameters for the classification algorithm of spreads


class spread_classify_description {
public:

   int f_projective_space;
   std::string projective_space_label;

   int f_starter_size;
   int starter_size;

   int f_k;
   int k;

   int f_poset_classification_control;
   std::string poset_classification_control_label;

   int f_output_prefix;
   std::string output_prefix;

   int f_recoordinatize;

   spread_classify_description();
   ~spread_classify_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// spread_classify.cpp
// ##############################################################################
```

```
//! to classify spreads of PG(k-1,q) in PG(n-1,q) where k divides n


class spread_classify {
public:

   spread_classify_description *Descr;
   geometry::spread_domain *SD;

   projective_geometry::projective_space_with_action *PA;
   groups::strong_generators *Strong_gens;

   groups::matrix_group *Mtx;

   long int block_size;
      // = r = {k choose 1}_q,
      // used in spread_lifting.spp


   int starter_size;
   int target_size; // = SD->spread_size


   actions::action *A;
      // P Gamma L(n,q)
   actions::action *A2;
      // action of A on grassmannian
      // of k-subspaces of V(n,q)
   induced_actions::action_on_grassmannian *AG;

   recoordinatize *R;
   poset_classification::classification_base_case *Base_case;

   // if f_recoordinatize is TRUE:
   long int *Starter;
   int Starter_size;
   groups::strong_generators *Starter_Strong_gens;

   poset_classification::poset_classification_control *Control;
   poset_classification::poset_with_group_action *Poset;
   poset_classification::poset_classification *gen;

   std::string prefix;

   apps_geometry::singer_cycle *Sing;
```

```
   // not used (commented out)

long int Nb;
   // Combi.generalized_binomial(n, k, q);
   // or R->nb_live_points if f_recoordinatize


isomorph::isomorph_worker *Worker;



spread_classify();
~spread_classify();
void init_basic(
      spread_classify_description *Descr,
      int verbose_level);
void init(
      geometry::spread_domain *SD,
      projective_geometry::projective_space_with_action *PA,
      int verbose_level);
void init2(int verbose_level);
void classify_partial_spreads(int verbose_level);
void lifting(
      int orbit_at_level, int level_of_candidates_file,
      int f_lexorder_test, int f_eliminate_graphs_if_possible,
      int &nb_vertices,
      //graph_theory::colored_graph *&CG,
      solvers::diophant *&Dio,
      long int *&col_labels,
      int &f_ruled_out,
      int verbose_level);
void setup_lifting(
      data_structures_groups::orbit_rep *R,
      std::string &output_prefix,
      solvers::diophant *&Dio, long int *&col_labels,
      int &f_ruled_out,
      int verbose_level);


// spread_classify2.cpp
void print_isomorphism_type(
      isomorph::isomorph *Iso,
   int iso_cnt, groups::sims *Stab, groups::schreier &Orb,
   long int *data, int verbose_level);
   // called from callback_print_isomorphism_type()
void print_isomorphism_type2(
      isomorph::isomorph *Iso,
```

```
        std::ostream &ost,
        int iso_cnt, groups::sims *Stab, groups::schreier &Orb,
        long int *data, int verbose_level);
   void save_klein_invariants(
        char *prefix,
      int iso_cnt,
      long int *data, int data_size, int verbose_level);
   void klein(
        std::ostream &ost,
        isomorph::isomorph *Iso,
      int iso_cnt, groups::sims *Stab, groups::schreier &Orb,
      long int *data, int data_size, int verbose_level);

   void report2(
        isomorph::isomorph &Iso, int verbose_level);
   void report3(
        isomorph::isomorph &Iso,
        std::ostream &ost, int verbose_level);
   void all_cooperstein_thas_quotients(
        isomorph::isomorph &Iso, int verbose_level);
   void cooperstein_thas_quotients(
        isomorph::isomorph &Iso, std::ofstream &f,
      int h, int &cnt, int verbose_level);
   void orbit_info_short(
        std::ostream &ost, isomorph::isomorph &Iso,
        int h, int verbose_level);
   void report_stabilizer(isomorph::isomorph &Iso,
        std::ostream &ost, int orbit,
      int verbose_level);
};




// ##############################################################################
// spread_create_description.cpp
// ##############################################################################

//! to describe the construction of a known spread from the command line

class spread_create_description {

public:

   int f_kernel_field;
   std::string kernel_field_label;
```

```
   int f_group;
   std::string group_label;

   int f_group_on_subspaces;
   std::string group_on_subspaces_label;

   int f_k;
   int k;

   int f_catalogue;
   int iso;

   int f_family;
   std::string family_name;

   int f_spread_set;
   std::string spread_set_label;
   std::string spread_set_label_tex;
   std::string spread_set_data;

   int f_transform;
   std::vector<std::string> transform_text;
   std::vector<int> transform_f_inv;



   spread_create_description();
   ~spread_create_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();
};

// ##############################################################################
// spread_create.cpp
// ##############################################################################

//! to create a known spread using a description from class spread_create_descrip
tion



class spread_create {

public:
   spread_create_description *Descr;
```

```
   std::string prefix;
   std::string label_txt;
   std::string label_tex;

   apps_algebra::any_group *G;
   apps_algebra::any_group *G_on_subspaces;

   int q;
   field_theory::finite_field *F;
   int k;

   int f_semilinear;

   actions::action *A;
   int degree;


   geometry::grassmann *Grass;

   long int *set;
   int sz;

   int f_has_group;
   groups::strong_generators *Sg;

   geometry::andre_construction *Andre;



   spread_create();
   ~spread_create();
   void init(spread_create_description *Descr,
         int verbose_level);
   void apply_transformations(
         std::vector<std::string> transform_coeffs,
         std::vector<int> f_inverse_transform, int verbose_level);
};



// ##############################################################################
// spread_lifting.cpp
// ##############################################################################

//! creates spreads from partial spreads using class exact_cover
```

```
class spread_lifting {
public:

   spread_classify *S;
   data_structures_groups::orbit_rep *R;
   std::string output_prefix;

   //long int *starter; // = R->rep
   //int starter_size; // = R->level
   //int starter_case_number; // = R->orbit_at_level
   //int starter_number_of_cases; // = R->nb_cases

   //long int *candidates; // = R->candidates
   //int nb_candidates;  // = R->nb_candidates

   //groups::strong_generators *Strong_gens; // = R->Strong_gens

   int f_lex;



   long int *points_covered_by_starter;
      // [nb_points_covered_by_starter]
   int nb_points_covered_by_starter;

   int nb_free_points;
   long int *free_point_list; // [nb_free_points]
   long int *point_idx; // [nb_points_total]
      // point_idx[i] = index of a point in free_point_list
      // or -1 if the point is in points_covered_by_starter


   int nb_colors;
   int *colors; // [nb_colors]


   int nb_needed;


   long int *reduced_candidates;
   int nb_reduced_candidates;

   int nb_cols;
   int *col_color; // [nb_cols]
   long int *col_labels; // [nb_cols]
```

```
   spread_lifting();
   ~spread_lifting();
   void init(spread_classify *S,
         data_structures_groups::orbit_rep *R,
      std::string &output_prefix,
      int f_lex,
      int verbose_level);
   void compute_points_covered_by_starter(
      int verbose_level);
   void prepare_free_points(
      int verbose_level);
   void print_free_points();
   void compute_colors(int &f_ruled_out, int verbose_level);
   void reduce_candidates(int verbose_level);
   solvers::diophant *create_system(int verbose_level);
   int is_e1_vector(int *v);
   int is_zero_vector(int *v);
   void create_graph(
         data_structures::bitvector *Adj,
         int verbose_level);
   void create_dummy_graph(int verbose_level);

};


// ##############################################################################
// spread_table_activity_description.cpp
// ##############################################################################

//! description of an activity for a spread table


class spread_table_activity_description {
public:


   int f_find_spread;
   std::string find_spread_text;

   int f_find_spread_and_dualize;
   std::string find_spread_and_dualize_text;

   int f_dualize_packing;
   std::string dualize_packing_text;

   int f_print_spreads;
   std::string print_spreads_idx_text;
```

```
   int f_export_spreads_to_csv;
   std::string export_spreads_to_csv_fname;
   std::string export_spreads_to_csv_idx_text;


   int f_find_spreads_containing_two_lines;
   int find_spreads_containing_two_lines_line1;
   int find_spreads_containing_two_lines_line2;

   int f_find_spreads_containing_one_line;
   int find_spreads_containing_one_line_line_idx;



   spread_table_activity_description();
   ~spread_table_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// spread_table_activity.cpp
// ##############################################################################

//! an activity for a spread table


class spread_table_activity {
public:

   spread_table_activity_description *Descr;
   packings::packing_classify *P;



   spread_table_activity();
   ~spread_table_activity();
   void init(
         spreads::spread_table_activity_description *Descr,
         packings::packing_classify *P,
         int verbose_level);
   void perform_activity(int verbose_level);
```

```
   void export_spreads_to_csv(
         std::string &fname,
         int *spread_idx, int nb, int verbose_level);
   void report_spreads(
         int *spread_idx, int nb, int verbose_level);
   void report_spread2(
         std::ostream &ost, int spread_idx, int verbose_level);

};


// ##############################################################################
// spread_table_with_selection.cpp
// ##############################################################################

//! spreads tables with a selection of isomorphism types


class spread_table_with_selection {
public:

   spread_classify *T;
   field_theory::finite_field *F;
   int q;
   int spread_size;
   int size_of_packing;
   int nb_lines;
   int f_select_spread;
   std::string select_spread_text;

   int *select_spread;
   int select_spread_nb;


   std::string path_to_spread_tables;


   long int *spread_reps; // [nb_spread_reps * T->spread_size]
   int *spread_reps_idx; // [nb_spread_reps]
   long int *spread_orbit_length; // [nb_spread_reps]
   int nb_spread_reps;
   long int total_nb_of_spreads; // = sum i :  spread_orbit_length[i]
   int nb_iso_types_of_spreads;
   // the number of spreads
   // from the classification
   int *sorted_packing;
   int *dual_packing;
```

```
geometry::spread_tables *Spread_tables;
int *tmp_isomorphism_type_of_spread; // for packing_swap_func

data_structures::bitvector *Bitvec;

actions::action *A_on_spreads;

spread_table_with_selection();
~spread_table_with_selection();
void init(spread_classify *T,
   int f_select_spread,
   std::string &select_spread_text,
   std::string &path_to_spread_tables,
   int verbose_level);
void compute_spread_table(int verbose_level);
void compute_spread_table_from_scratch(int verbose_level);
void create_action_on_spreads(int verbose_level);
int find_spread(long int *set, int verbose_level);
long int *get_spread(int spread_idx);
void find_spreads_containing_two_lines(
      std::vector<int> &v,
      int line1, int line2, int verbose_level);
int test_if_packing_is_self_dual(
      int *packing, int verbose_level);
void predict_spread_table_length(
      actions::action *A,
      groups::strong_generators *Strong_gens,
   int verbose_level);
void make_spread_table(
      actions::action *A, actions::action *A2,
      groups::strong_generators *Strong_gens,
      long int **&Sets, int *&Prev,
      int *&Label, int *&first, int *&len,
      int *&isomorphism_type_of_spread,
      int verbose_level);
void compute_covered_points(
   long int *&points_covered_by_starter,
   int &nb_points_covered_by_starter,
   long int *starter, int starter_size,
   int verbose_level);
// points_covered_by_starter are the lines that
// are contained in the spreads chosen for the starter
void compute_free_points2(
   long int *&free_points2,
   int &nb_free_points2, long int *&free_point_idx,
   long int *points_covered_by_starter,
```

```
      int nb_points_covered_by_starter,
      long int *starter, int starter_size,
      int verbose_level);
   // free_points2 are actually the free lines,
   // i.e., the lines that are not
   // yet part of the partial packing
   void compute_live_blocks2(
         solvers_package::exact_cover *EC,
         int starter_case,
      long int *&live_blocks2, int &nb_live_blocks2,
      long int *points_covered_by_starter,
      int nb_points_covered_by_starter,
      long int *starter, int starter_size,
      int verbose_level);
   void compute_adjacency_matrix(int verbose_level);
   int is_adjacent(int i, int j);

};

// ##############################################################################
// translation_plane_activity_description.cpp
// ##############################################################################

//! description of an activity regarding a translation plane



class translation_plane_activity_description {

public:

   int f_export_incma;

   int f_p_rank;
   int p_rank_p;

   int f_report;

   translation_plane_activity_description();
   ~translation_plane_activity_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();

};
```

```
// ##############################################################################
// translation_plane_activity.cpp
// ##############################################################################

//! an activity regarding a translation plane



class translation_plane_activity {

public:

   translation_plane_activity_description *Descr;
   data_structures_groups::translation_plane_via_andre_model *TP;


   translation_plane_activity();
   ~translation_plane_activity();
   void init(
         translation_plane_activity_description *Descr,
         data_structures_groups::translation_plane_via_andre_model *TP,
         int verbose_level);
   void perform_activity(int verbose_level);

};









}}}



#endif /* SRC_LIB_TOP_LEVEL_SPREADS_SPREADS_H_ */
```

7.13 Quartic Curves

```
/*
 * quartic_curves.cpp
 *
 *  Created on: May 25, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_QUARTIC_CURVES_QUARTIC_CURVES_CPP_
#define SRC_LIB_TOP_LEVEL_QUARTIC_CURVES_QUARTIC_CURVES_CPP_




namespace orbiter {
namespace layer5_applications {
namespace applications_in_algebraic_geometry {
namespace quartic_curves {


// ###############################################################################
// quartic_curve_activity_description.cpp
// ###############################################################################

//! description of an activity associated with a quartic curve


class quartic_curve_activity_description {
public:

   int f_report;

   int f_export_something;
   std::string export_something_what;


   //int f_export_points;

   int f_create_surface;

   int f_extract_orbit_on_bitangents_by_length;
   int extract_orbit_on_bitangents_by_length_length;

   int f_extract_specific_orbit_on_bitangents_by_length;
   int extract_specific_orbit_on_bitangents_by_length_length;
   int extract_specific_orbit_on_bitangents_by_length_index;
```

```
   int f_extract_specific_orbit_on_kovalevski_points_by_length;
   int extract_specific_orbit_on_kovalevski_points_by_length_length;
   int extract_specific_orbit_on_kovalevski_points_by_length_index;


   quartic_curve_activity_description();
   ~quartic_curve_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();

};


// ##############################################################################
// quartic_curve_activity.cpp
// ##############################################################################

//! an activity associated with a quartic curve


class quartic_curve_activity {
public:

   quartic_curve_activity_description *Descr;
   quartic_curve_create *QC;


   quartic_curve_activity();
   ~quartic_curve_activity();
   void init(
         quartic_curve_activity_description
            *Quartic_curve_activity_description,
         quartic_curve_create *QC, int verbose_level);
   void perform_activity(
         int verbose_level);
   void do_report(
         quartic_curve_create *QC,
         int verbose_level);

};




// ##############################################################################
```

```
// quartic_curve_create_description.cpp
// ##############################################################################



//! to describe a quartic curve from the command line


class quartic_curve_create_description {

public:

   int f_space;
   std::string space_label;

   int f_space_pointer;
   projective_geometry::projective_space_with_action
      *space_pointer;

   int f_label_txt;
   std::string label_txt;

   int f_label_tex;
   std::string label_tex;

   int f_label_for_summary;
   std::string label_for_summary;

   int f_catalogue;
   int iso;

   int f_by_coefficients;
   std::string coefficients_text;


   int f_by_equation;
   std::string equation_name_of_formula;
   std::string equation_name_of_formula_tex;
   std::string equation_managed_variables;
   std::string equation_text;
   std::string equation_parameters;
   std::string equation_parameters_tex;

   int f_from_cubic_surface;
   std::string from_cubic_surface_label;
   int from_cubic_surface_point_orbit_idx;
```

```
   int f_override_group;
   std::string override_group_order;
   int override_group_nb_gens;
   std::string override_group_gens;

   std::vector<std::string> transform_coeffs;
   std::vector<int> f_inverse_transform;



   quartic_curve_create_description();
   ~quartic_curve_create_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();
   //int get_q();
};



// ##############################################################################
// quartic_curve_create.cpp
// ##############################################################################


//! to create a quartic curve from a description using class quartic_curve_create
_description


class quartic_curve_create {

public:
   quartic_curve_create_description *Descr;

   std::string prefix;
   std::string label_txt;
   std::string label_tex;

   int f_ownership;

   int q;
   field_theory::finite_field *F;

   int f_semilinear;
```

```
projective_geometry::projective_space_with_action *PA;

quartic_curve_domain_with_action *QCDA;

algebraic_geometry::quartic_curve_object *QO;

quartic_curve_object_with_action *QOA;

int f_has_group;
groups::strong_generators *Sg;
int f_has_nice_gens;
data_structures_groups::vector_ge *nice_gens;

int f_has_quartic_curve_from_surface;
quartic_curves::quartic_curve_from_surface *QC_from_surface;



quartic_curve_create();
~quartic_curve_create();
void create_quartic_curve(
      quartic_curve_create_description
         *Quartic_curve_descr,
      int verbose_level);
void init_with_data(
      quartic_curve_create_description
         *Descr,
      projective_geometry::projective_space_with_action
         *PA,
      int verbose_level);
void init(
      quartic_curve_create_description *Descr,
      projective_geometry::projective_space_with_action
         *PA,
      int verbose_level);
void create_quartic_curve_from_description(
      quartic_curve_domain_with_action *DomA,
      int verbose_level);
void override_group(
      std::string &group_order_text,
      int nb_gens,
      std::string &gens_text,
      int verbose_level);
void create_quartic_curve_by_coefficients(
      std::string &coefficients_text,
      int verbose_level);
void create_quartic_curve_by_coefficient_vector(
```

```
        int *eqn15,
        int verbose_level);
   void create_quartic_curve_from_catalogue(
        quartic_curve_domain_with_action *DomA,
        int iso,
        int verbose_level);
   void create_quartic_curve_by_equation(
        std::string &name_of_formula,
        std::string &name_of_formula_tex,
        std::string &managed_variables,
        std::string &equation_text,
        std::string &equation_parameters,
        std::string &equation_parameters_tex,
        int verbose_level);
   void create_quartic_curve_from_cubic_surface(
        std::string &cubic_surface_label,
        int pt_orbit_idx,
        //int f_TDO,
        int verbose_level);
   void apply_transformations(
      std::vector<std::string> &transform_coeffs,
      std::vector<int> &f_inverse_transform,
      int verbose_level);
   void apply_single_transformation(int f_inverse,
        int *transformation_coeffs,
        int sz, int verbose_level);
   void compute_group(
        projective_geometry::projective_space_with_action *PA,
        int verbose_level);
   // ToDo
   void report(
        std::ostream &ost, int verbose_level);
   void print_general(
        std::ostream &ost, int verbose_level);
   void export_something(
        std::string &what, int verbose_level);

};


// ##############################################################################
// quartic_curve_domain_with_action.cpp
// ##############################################################################

//! a domain for quartic curves in projective space with group action
```

```
class quartic_curve_domain_with_action {

public:


   projective_geometry::projective_space_with_action *PA;

   int f_semilinear;

   algebraic_geometry::quartic_curve_domain *Dom; // do not free

   actions::action *A; // linear group PGGL(3,q)


   actions::action *A_on_lines; // linear group PGGL(3,q) acting on lines

   int *Elt1;

   induced_actions::action_on_homogeneous_polynomials *AonHPD_4_3;


   quartic_curve_domain_with_action();
   ~quartic_curve_domain_with_action();
   void init(
         algebraic_geometry::quartic_curve_domain *Dom,
         projective_geometry::projective_space_with_action *PA,
         int verbose_level);

};


// ##############################################################################
// quartic_curve_from_surface.cpp
// ##############################################################################


//! construction of a quartic curve from a cubic surface by means of projecting t
he intersection with the tangent cone at a point


class quartic_curve_from_surface {

public:

   std::string label;
   std::string label_tex;
```

```
int f_has_SC;
cubic_surfaces_in_general::surface_create *SC;

cubic_surfaces_in_general::surface_object_with_action *SOA;

int pt_orbit;
int equation_nice[20]; // equation after transformation
int *transporter;
   // the transformation that maps
   // the point off the lines to (1,0,0,0)

int v[4]; // = (1,0,0,0)
int pt_A; // = SOA->SO->SOP->Pts_not_on_lines[i];
int pt_B; // = SOA->Surf->rank_point(v);

long int *Lines_nice; // surface lines after transformation
int nb_lines;

long int *Bitangents;
int nb_bitangents; // = nb_lines + 1


// computed by split_nice_equation starting
// from the equation of the cubic surface:
int *f1; // terms involving X0^2, with X0^2 removed (linear)
int *f2; // terms involving X0, with X0 removed (quadratic)
int *f3; // terms free of X0 (cubic)

long int *Pts_on_surface; // points on the transformed cubic surface
int nb_pts_on_surface;

// the equation of the quartic curve:
int *curve; // poly1 + poly2 = f2^2 - 4 * f1 * f3
int *poly1; // f2 * f2
int *poly2; // -4 * f1 * f3
int two, four, mfour; // 2, 4, -4 in F

int *tangent_quadric; // = 2 * x_0 * f_1 + f_2
long int *Pts_on_tangent_quadric;
   // = SOA->Surf->Poly2_4->enumerate_points(tangent_quadric)
int nb_pts_on_tangent_quadric;

//int *line_type;
//int *type_collected;
```

```
   //int *Class_pts;
   //int nb_class_pts;
   //long int *Pts_intersection;
   //int nb_pts_intersection;

   long int *Pts_on_curve;
      // = SOA->Surf->Poly4_x123->enumerate_points(curve)
   int sz_curve;

#if 0
   strong_generators *gens_copy;
   set_and_stabilizer *moved_surface;
   //strong_generators *stab_gens_moved_surface;
   strong_generators *stab_gens_P0;
#endif

   groups::strong_generators *Stab_gens_quartic;



   quartic_curve_from_surface();
   ~quartic_curve_from_surface();
   void init(
         cubic_surfaces_in_general::surface_object_with_action *SOA,
         int verbose_level);
   void init_surface_create(
         cubic_surfaces_in_general::surface_create *SC,
         int verbose_level);
   void init_labels(
         std::string &label, std::string &label_tex,
         int verbose_level);
   void quartic(
         int pt_orbit, int verbose_level);
   void compute_quartic(
         int pt_orbit,
      int *equation, long int *Lines, int nb_lines,
      int verbose_level);
   void compute_stabilizer(
         int verbose_level);
   void cheat_sheet_quartic_curve(
         std::ostream &ost,
         int f_TDO,
         int verbose_level);

};
```

```
// ##############################################################################
// quartic_curve_object_with_action.cpp
// ##############################################################################


//! an instance of a quartic curve together with its stabilizer


class quartic_curve_object_with_action {

public:

   field_theory::finite_field *F; // do not free

   quartic_curve_domain_with_action *DomA;

   algebraic_geometry::quartic_curve_object *QO; // do not free
   groups::strong_generators *Aut_gens;
      // generators for the automorphism group

   int f_has_nice_gens;
   data_structures_groups::vector_ge *nice_gens;

   groups::strong_generators *projectivity_group_gens;
   groups::sylow_structure *Syl;

   actions::action *A_on_points;

   groups::schreier *Orbits_on_points;

   quartic_curve_object_with_action();
   ~quartic_curve_object_with_action();
   void init(
         quartic_curve_domain_with_action *DomA,
         algebraic_geometry::quartic_curve_object *QO,
         groups::strong_generators *Aut_gens,
         int verbose_level);
   void export_something(
         std::string &what,
         std::string &fname_base, int verbose_level);

};
```

```
}}}}


#endif /* SRC_LIB_TOP_LEVEL_QUARTIC_CURVES_QUARTIC_CURVES_CPP_ */
```

7.14 Cubic Surfaces and Arcs

```
/*
 * surfaces_and_arcs.h
 *
 *  Created on: Jun 26, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_SURFACES_SURFACES_AND_ARCS_SURFACES_AND_ARCS_H_
#define SRC_LIB_TOP_LEVEL_SURFACES_SURFACES_AND_ARCS_SURFACES_AND_ARCS_H_

namespace orbiter {
namespace layer5_applications {
namespace applications_in_algebraic_geometry {
namespace cubic_surfaces_and_arcs {


// ##############################################################################
// arc_lifting.cpp
// ##############################################################################

//! creates a cubic surface from a 6-arc in a plane using trihedral pairs


class arc_lifting {

public:

   int q;
   field_theory::finite_field *F;

   algebraic_geometry::surface_domain *Surf;

   cubic_surfaces_in_general::surface_with_action *Surf_A;

   long int *arc;
   int arc_size;





   int *the_equation; // [20]

   algebraic_geometry::web_of_cubic_curves *Web;
```

```
   trihedral_pair_with_action *Trihedral_pair;

   arc_lifting();
   ~arc_lifting();
   void create_surface_and_group(
         cubic_surfaces_in_general::surface_with_action *Surf_A,
         long int *Arc6,
      int verbose_level);
   void create_web_of_cubic_curves(int verbose_level);
   void report(std::ostream &ost, int verbose_level);
   void report_equation(std::ostream &ost);
};


// ##############################################################################
// arc_orbits_on_pairs.cpp
// ##############################################################################

//! orbits on pairs of points of a non-conical six-arc in PG(2,q)


class arc_orbits_on_pairs {
public:

   surfaces_arc_lifting *SAL;

   actions::action *A; // this is the 3x3 group

   data_structures_groups::set_and_stabilizer *The_arc;
   actions::action *A_on_arc;

   int arc_idx;
   poset_classification::poset_with_group_action *Poset;
   poset_classification::poset_classification_control *Control;
   poset_classification::poset_classification *Orbits_on_pairs;

   int nb_orbits_on_pairs;

   arc_partition *Table_orbits_on_partition; // [nb_orbits_on_pairs]

   int total_nb_orbits_on_partitions;

   int *partition_orbit_first; // [nb_orbits_on_pairs]
   int *partition_orbit_len; // [nb_orbits_on_pairs]


   arc_orbits_on_pairs();
```

```
   ~arc_orbits_on_pairs();
   void init(
      surfaces_arc_lifting *SAL, int arc_idx,
      actions::action *A,
      int verbose_level);
   void print();
   void recognize(long int *pair, int *transporter,
         int &orbit_idx, int verbose_level);

};


// ##############################################################################
// arc_partition.cpp
// ##############################################################################

//! orbits on the partitions of the remaining four points of a non-conical arc


class arc_partition {
public:

   arc_orbits_on_pairs *OP;

   actions::action *A;
   actions::action *A_on_arc;

   int pair_orbit_idx;
   data_structures_groups::set_and_stabilizer *The_pair;

   long int arc_remainder[4];

   actions::action *A_on_rest;
   actions::action *A_on_partition;

   groups::schreier *Orbits_on_partition;

   int nb_orbits_on_partition;


   arc_partition();
   ~arc_partition();
   void init(
      arc_orbits_on_pairs *OP, int pair_orbit_idx,
      actions::action *A, actions::action *A_on_arc,
      int verbose_level);
   void recognize(int *partition, int *transporter,
```

```
        int &orbit_idx, int verbose_level);

};


// ##############################################################################
// classify_trihedral_pairs.cpp
// ##############################################################################


//! classification of double triplets in PG(3,q)


class classify_trihedral_pairs {

public:

   int q;
   field_theory::finite_field *F;
   actions::action *A;

   cubic_surfaces_in_general::surface_with_action *Surf_A;
   algebraic_geometry::surface_domain *Surf;

   groups::strong_generators *gens_type1;
   groups::strong_generators *gens_type2;

   poset_classification::poset_with_group_action *Poset1;
   poset_classification::poset_with_group_action *Poset2;
   poset_classification::poset_classification *orbits_on_trihedra_type1;
   poset_classification::poset_classification *orbits_on_trihedra_type2;

   int nb_orbits_type1;
   int nb_orbits_type2;
   int nb_orbits_ordered_total;

   invariant_relations::flag_orbits *Flag_orbits;

   int nb_orbits_trihedral_pairs;

   invariant_relations::classification_step *Trihedral_pairs;



   classify_trihedral_pairs();
   ~classify_trihedral_pairs();
   void init(
```

```
        cubic_surfaces_in_general::surface_with_action *Surf_A,
        int verbose_level);

    void classify_orbits_on_trihedra(
        poset_classification::poset_classification_control *Control1,
        poset_classification::poset_classification_control *Control2,
        int verbose_level);
    void report_summary(std::ostream &ost);
    void report(std::ostream &ost);
    void list_orbits_on_trihedra_type1(
        std::ostream &ost, int f_detailed);
    void list_orbits_on_trihedra_type2(
        std::ostream &ost, int f_detailed);
    void early_test_func_type1(
        long int *S, int len,
        long int *candidates, int nb_candidates,
        long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
    void early_test_func_type2(
        long int *S, int len,
        long int *candidates, int nb_candidates,
        long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
    void identify_three_planes(
        int p1, int p2, int p3,
      int &type, int *transporter, int verbose_level);
    void classify(
        poset_classification::poset_classification_control *Control1,
        poset_classification::poset_classification_control *Control2,
        int verbose_level);
    void downstep(int verbose_level);
    void upstep(int verbose_level);
    void print_trihedral_pairs_summary(
        std::ostream &ost);
    void print_trihedral_pairs(
        std::ostream &ost,
      int f_with_stabilizers);
    groups::strong_generators *identify_trihedral_pair_and_get_stabilizer(
      long int *planes6, int *transporter, int &orbit_index,
      int verbose_level);
    void identify_trihedral_pair(
        long int *planes6,
      int *transporter, int &orbit_index, int verbose_level);

};
```

```
// ##############################################################################
// six_arcs_not_on_a_conic.cpp
// ##############################################################################

//! classification of six-arcs not on a conic in PG(2,q)


class six_arcs_not_on_a_conic {

public:

   apps_geometry::arc_generator_description *Descr;
   projective_geometry::projective_space_with_action *PA;

   apps_geometry::arc_generator *Gen;

   int nb_orbits;

   int *Not_on_conic_idx;
   int nb_arcs_not_on_conic;

   six_arcs_not_on_a_conic();
   ~six_arcs_not_on_a_conic();
   void init(
         apps_geometry::arc_generator_description *Descr,
         projective_geometry::projective_space_with_action *PA,
      int f_test_nb_Eckardt_points, int nb_E,
      int verbose_level);
   void recognize(
         long int *arc6, int *transporter,
         int &orbit_not_on_conic_idx, int verbose_level);
   void report_latex(
         std::ostream &ost);
   void report_specific_arc_basic(
         std::ostream &ost, int arc_idx);
   void report_specific_arc(
         std::ostream &ost, int arc_idx);
};



// ##############################################################################
// surface_classify_using_arc.cpp
// ##############################################################################
```

```
//! classification of cubic surfaces using non-conical six-arcs as substructures


class surface_classify_using_arc {
public:

   cubic_surfaces_in_general::surface_with_action *Surf_A;



   six_arcs_not_on_a_conic *Six_arcs;
   apps_geometry::arc_generator_description *Descr;

   int *transporter;


   int nb_surfaces;
   surface_create_by_arc_lifting *SCAL;
   // allocated as [Six_arcs->nb_arcs_not_on_conic], used as [nb_surfaces]

   int *Arc_identify_nb;
   int *Arc_identify;  // [Six_arcs->nb_arcs_not_on_conic]
   //[Six_arcs->nb_arcs_not_on_conic * Six_arcs->nb_arcs_not_on_conic]
   int *f_deleted; // [Six_arcs->nb_arcs_not_on_conic]

   int *Decomp; // [Six_arcs->nb_arcs_not_on_conic * nb_surfaces]

   surface_classify_using_arc();
   ~surface_classify_using_arc();
   void classify_surfaces_through_arcs_and_trihedral_pairs(
         std::string &Control_six_arcs_label,
         cubic_surfaces_in_general::surface_with_action *Surf_A,
         int f_test_nb_Eckardt_points, int nb_E,
         int verbose_level);
   void report(
         graphics::layered_graph_draw_options *Opt,
         int verbose_level);
   void report2(std::ostream &ost,
         graphics::layered_graph_draw_options *Opt,
         int verbose_level);
   void report_decomposition_matrix(std::ostream &ost, int verbose_level);
};

// ##############################################################################
// surface_create_by_arc_lifting.cpp
// ##############################################################################
```

```
//! to create a single cubic surface from an arc using arc lifting


class surface_create_by_arc_lifting {
public:

   surface_classify_using_arc *SCA;

   int arc_idx;
   int surface_idx;
   long int *Arc6;


   arc_lifting *AL;

   cubic_surfaces_in_general::surface_object_with_action *SOA;


   int nine_lines_idx[9];
   std::string arc_label;
   std::string arc_label_short;

   cubic_surfaces_in_general::surface_clebsch_map *Clebsch; // [SOA->Orbits_on_sin
gle_sixes->nb_orbits]
   int *Other_arc_idx; // [SOA->Orbits_on_single_sixes->nb_orbits]

   surface_create_by_arc_lifting();
   ~surface_create_by_arc_lifting();
   void init(int arc_idx,
         surface_classify_using_arc *SCA, int verbose_level);
   void report_summary(std::ostream &ost, int verbose_level);
   void report(std::ostream &ost,
         graphics::layered_graph_draw_options *Opt,
         int verbose_level);

};


// ##############################################################################
// surfaces_arc_lifting_definition_node.cpp
// ##############################################################################



//! flag orbit node which is a definition node and hence describes a surface
```

```
class surfaces_arc_lifting_definition_node {
public:

   surfaces_arc_lifting *Lift;

   int f;
   int orbit_idx;

   algebraic_geometry::surface_object *SO;
   cubic_surfaces_in_general::surface_object_with_action *SOA;



   groups::strong_generators *Flag_stab_gens;
   ring_theory::longinteger_object Flag_stab_go;


   int three_lines_idx[45 * 3];
      // the index into Lines[] of the
      // three lines in the chosen tritangent plane
      // This is computed from the Schlaefli labeling
      // using the eckardt point class.

   long int three_lines[45 * 3];
      // the three lines in the chosen tritangent plane



   algebraic_geometry::seventytwo_cases Seventytwo[45 * 72];
      // for each of the 45 tritangent planes,
      // there are 72 Clebsch maps


   int nb_coset_reps;
   surfaces_arc_lifting_trace **T; // [nb_coset_reps]
   data_structures_groups::vector_ge *coset_reps;

   int *relative_order_table; // [nb_coset_reps]

   int f_has_F2;
   int *F2; // F2[i] = Seventytwo[i].f2;
   data_structures::tally *tally_F2;


   surfaces_arc_lifting_definition_node();
   ~surfaces_arc_lifting_definition_node();
   void init_with_27_lines(surfaces_arc_lifting *Lift,
```

```
        int f, int orbit_idx, long int *Lines27, int *eqn20,
        int verbose_level);
   void tally_f2(int verbose_level);
   void report(int verbose_level);
   void report2(std::ostream &ost, int verbose_level);
   void report_cosets(std::ostream &ost, int verbose_level);
   void report_cosets_detailed(std::ostream &ost, int verbose_level);
   void report_cosets_HDS(std::ostream &ost, int verbose_level);
   void report_HDS_top(std::ostream &ost);
   void report_HDS_bottom(std::ostream &ost);
   void report_cosets_T3(std::ostream &ost, int verbose_level);
   void report_T3_top(std::ostream &ost);
   void report_T3_bottom(std::ostream &ost);
   void report_tally_F2(std::ostream &ost, int verbose_level);
   void report_Clebsch_maps(std::ostream &ost, int verbose_level);
   void report_Clebsch_maps_for_one_tritangent_plane(std::ostream &ost,
        int plane_idx, int verbose_level);
};


// ##############################################################################
// surfaces_arc_lifting_trace.cpp
// ##############################################################################



//! tracing data to be used during the classification of cubic surfaces using lif
ted 6-arcs


class surfaces_arc_lifting_trace {
public:

   surfaces_arc_lifting_upstep *Up;

   int f, f2;

   int po, so;

   // po = Lift->Flag_orbits->Flag_orbit_node[f].downstep_primary_orbit;
   // so = Lift->Flag_orbits->Flag_orbit_node[f].downstep_secondary_orbit;

   // 3x3 matrices of elements in PGGL(3,q)
   int *Elt_alpha2;
      // Using local coordinates P6_local[6],
      // maps the arc P6[6] to the canonical arc in the classification.
   int *Elt_beta1;
```

```
   // Moves the arc points on m1 to P1 and P2.
   // Computed using
   // Table_orbits_on_pairs[orbit_not_on_conic_idx].recognize
int *Elt_beta2;
   // Moves the partition, computed using
   // Table_orbits_on_partition[pair_orbit_idx].recognize

// temporary matrices
int *Elt_T1;
int *Elt_T2;
int *Elt_T3;
int *Elt_T4;

// 4x4 matrices or elements in PGGL(4,q):
int *Elt_Alpha1;
   // Moves the chosen tritangent plane to the hyperplane X3=0

int *Elt_Alpha2; // embedding of alpha2
int *Elt_Beta1; // embedding of beta1
int *Elt_Beta2; // embedding of beta2
int *Elt_Beta3;
   // Moves the two lines  which are the images of l1 and l2
   // under the group elements computed so far
   // to the canonical ones associated with the flag f2.
   // Computed with hyperplane_lifting_with_two_lines_moved

   // if f2 = f then
   // Alpha1 * Alpha2 * Beta1 * Beta2 * Beta3
   // is an automorphism of the surface


int upstep_idx;



int seventytwo_case_idx;

algebraic_geometry::seventytwo_cases The_case;


surfaces_arc_lifting_trace();
~surfaces_arc_lifting_trace();
void init(surfaces_arc_lifting_upstep *Up,
      int seventytwo_case_idx, int verbose_level);
void process_flag_orbit(surfaces_arc_lifting_upstep *Up, int verbose_level);
void move_arc(int verbose_level);
void move_plane_and_arc(long int *P6a, int verbose_level);
```

```
   void make_arc_canonical(
         long int *P6_local, long int *P6_local_canonical,
         int &orbit_not_on_conic_idx, int verbose_level);
   void compute_beta1(algebraic_geometry::seventytwo_cases *The_case, int verbose_
level);
   void compute_beta2(int orbit_not_on_conic_idx,
         int pair_orbit_idx, int &partition_orbit_idx,
         int *the_partition4, int verbose_level);
   void lift_group_elements_and_move_two_lines(int verbose_level);
   void embed(int *Elt_A3, int *Elt_A4, int verbose_level);
   void report_product(std::ostream &ost, int *Elt, int verbose_level);

};



// ##############################################################################
// surfaces_arc_lifting_upstep.cpp
// ##############################################################################




//! classification of cubic surfaces using lifted 6-arcs


class surfaces_arc_lifting_upstep {
public:

   surfaces_arc_lifting *Lift;

   int *f_processed; // [Lift->Flag_orbits->nb_flag_orbits]
   int nb_processed;

   int pt_representation_sz;
   long int *Flag_representation;
   long int *Flag2_representation;
      // used only in upstep_group_elements

   ring_theory::longinteger_object A4_go;


   double progress;
   long int Lines[27];
   int eqn20[20];
```

```
   surfaces_arc_lifting_definition_node *D;

   //vector_ge *coset_reps;
   //int nb_coset_reps;

   //strong_generators *Flag_stab_gens;
   //longinteger_object Flag_stab_go;

   int f;

   int tritangent_plane_idx;
      // the tritangent plane picked for the Clebsch map,
      // using the Schlaefli labeling, in [0,44].


   int three_lines_idx[3];
      // the index into Lines[] of the
      // three lines in the chosen tritangent plane
      // This is computed from the Schlaefli labeling
      // using the eckardt point class.

   long int three_lines[3];
      // the three lines in the chosen tritangent plane



   algebraic_geometry::seventytwo_cases Seventytwo[72];

   int seventytwo_case_idx;


   surfaces_arc_lifting_upstep();
   ~surfaces_arc_lifting_upstep();
   void init(surfaces_arc_lifting *Lift, int verbose_level);
   void process_flag_orbit(int verbose_level);
   void compute_stabilizer(surfaces_arc_lifting_definition_node *D,
         groups::strong_generators *&Aut_gens, int verbose_level);
   void process_tritangent_plane(surfaces_arc_lifting_definition_node *D,
         int verbose_level);
   void make_seventytwo_cases(int verbose_level);

};
```

```
// ##############################################################################
// surfaces_arc_lifting.cpp
// ##############################################################################

//! classification of cubic surfaces using lifted 6-arcs


class surfaces_arc_lifting {
public:
   field_theory::finite_field *F;
   int q;
   groups::linear_group *LG4; // PGL(4,q)

   int f_semilinear;

   std::string fname_base;

   actions::action *A4; // the action of PGL(4,q) on points
   actions::action *A3; // the action of PGL(3,q) on points

   algebraic_geometry::surface_domain *Surf;
   cubic_surfaces_in_general::surface_with_action *Surf_A;

   six_arcs_not_on_a_conic *Six_arcs;

   arc_orbits_on_pairs *Table_orbits_on_pairs;
      // [Six_arcs->nb_arcs_not_on_conic]

   int nb_flag_orbits;

   // classification of surfaces:
   invariant_relations::flag_orbits *Flag_orbits;

   int *flag_orbit_fst; // [Six_arcs->nb_arcs_not_on_conic]
   int *flag_orbit_len; // [Six_arcs->nb_arcs_not_on_conic]

   int *flag_orbit_on_arcs_not_on_a_conic_idx; // [Flag_orbits->nb_flag_orbits]
   int *flag_orbit_on_pairs_idx; // [Flag_orbits->nb_flag_orbits]
   int *flag_orbit_on_partition_idx; // [Flag_orbits->nb_flag_orbits]

   invariant_relations::classification_step *Surfaces;

   surfaces_arc_lifting();
   ~surfaces_arc_lifting();
   void init(
         cubic_surfaces_in_general::surface_with_action *Surf_A,
         std::string &Control_six_arcs_label,
```

```
      int f_test_nb_Eckardt_points, int nb_E,
      int verbose_level);
   void downstep(int verbose_level);
   void downstep_one_arc(int arc_idx,
         int &cur_flag_orbit, long int *Flag, int verbose_level);
   void report(
         std::string &Control_six_arcs_label,
         int verbose_level);
   void report2(std::ostream &ost,
         graphics::layered_graph_draw_options *draw_options,
         int verbose_level);
   void report_flag_orbits(std::ostream &ost, int verbose_level);
   void report_flag_orbits_in_detail(std::ostream &ost, int verbose_level);
   void report_surfaces_in_detail(std::ostream &ost, int verbose_level);
};



// ##############################################################################
// trihedral_pair_with_action.cpp
// ##############################################################################

//! a trihedral pair and its stabilizer


class trihedral_pair_with_action {

public:

   arc_lifting *AL;

   int The_six_plane_equations[6 * 4]; // [6 * 4]
   int *The_surface_equations; // [(q + 1) * 20]
   long int plane6_by_dual_ranks[6];
   int lambda, lambda_rk;
   int t_idx;

   groups::strong_generators *stab_gens_trihedral_pair; // stabilizer of trihedral
 pair
   groups::strong_generators *gens_subgroup;
   ring_theory::longinteger_object stabilizer_of_trihedral_pair_go;
   actions::action *A_on_equations;
   groups::schreier *Orb;
   ring_theory::longinteger_object stab_order;
   int trihedral_pair_orbit_index;
   data_structures_groups::vector_ge *cosets;
```

```
data_structures_groups::vector_ge *coset_reps;
long int nine_lines[9];
int *aut_T_index;
int *aut_coset_index;
groups::strong_generators *Aut_gens;


int F_plane[3 * 4];
int G_plane[3 * 4];
int *System; // [3 * 4 * 3]
//int nine_lines[9];

int Iso_type_as_double_triplet[120];
data_structures::tally *Double_triplet_type_distribution;
data_structures::set_of_sets *Double_triplet_types;
int *Double_triplet_type_values;
int nb_double_triplet_types;

int *transporter0;
int *transporter;
int *Elt1;
int *Elt2;
int *Elt3;
int *Elt4;
int *Elt5;


trihedral_pair_with_action();
~trihedral_pair_with_action();
void init(arc_lifting *AL, int verbose_level);
void loop_over_trihedral_pairs(
      data_structures_groups::vector_ge *cosets,
      data_structures_groups::vector_ge *&coset_reps,
   int *&aut_T_index, int *&aut_coset_index, int verbose_level);
void create_the_six_plane_equations(int t_idx,
   int verbose_level);
void create_surface_from_trihedral_pair_and_arc(
   int t_idx,
   int verbose_level);
   // plane6[6]
   // The_six_plane_equations[6 * 4]
   // The_surface_equations[(q + 1) * 20]
groups::strong_generators *create_stabilizer_of_trihedral_pair(
      int &trihedral_pair_orbit_index, int verbose_level);
void create_action_on_equations_and_compute_orbits(
   int *The_surface_equations,
   groups::strong_generators *gens_for_stabilizer_of_trihedral_pair,
```

```
      actions::action *&A_on_equations, groups::schreier *&Orb,
      int verbose_level);
   void create_clebsch_system(int verbose_level);
   void compute_iso_types_as_double_triplets(int verbose_level);
   void print_FG(std::ostream &ost);
   void print_equations();
   void report(std::ostream &ost, int verbose_level);
   void report_iso_type_as_double_triplets(std::ostream &ost);

};




}}}}




#endif /* SRC_LIB_TOP_LEVEL_SURFACES_SURFACES_AND_ARCS_SURFACES_AND_ARCS_H_ */
```

7.15 Cubic Surfaces and Double Sixes

```
/*
 * surfaces_and_double_sixes.h
 *
 *  Created on: Jun 26, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_SURFACES_SURFACES_AND_DOUBLE_SIXES_SURFACES_AND_DOUBLE_
SIXES_H_
#define SRC_LIB_TOP_LEVEL_SURFACES_SURFACES_AND_DOUBLE_SIXES_SURFACES_AND_DOUBLE_
SIXES_H_


namespace orbiter {
namespace layer5_applications {
namespace applications_in_algebraic_geometry {
namespace cubic_surfaces_and_double_sixes {



// ##############################################################################
// classification_of_cubic_surfaces_with_double_sixes_activity_description.cpp
// ##############################################################################

//! description of an activity for a classification of cubic surfaces with 27 lin
es with double sixes


class classification_of_cubic_surfaces_with_double_sixes_activity_description {
public:

   int f_report;
   poset_classification::poset_classification_report_options
      *report_options;

   int f_identify_Eckardt;

   int f_identify_F13;

   int f_identify_Bes;

   int f_identify_general_abcd;

   int f_isomorphism_testing;
   cubic_surfaces_in_general::surface_create_description
      *isomorphism_testing_surface1;
```

```
   cubic_surfaces_in_general::surface_create_description
      *isomorphism_testing_surface2;

   int f_recognize;
   cubic_surfaces_in_general::surface_create_description
      *recognize_surface;

   int f_create_source_code;

   int f_sweep_Cayley;


   classification_of_cubic_surfaces_with_double_sixes_activity_description();
   ~classification_of_cubic_surfaces_with_double_sixes_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();


};

// ##############################################################################
// classification_of_cubic_surfaces_with_double_sixes_activity.cpp
// ##############################################################################

//! an activity for a classification of cubic surfaces with 27 lines with double
sixes


class classification_of_cubic_surfaces_with_double_sixes_activity {
public:

   classification_of_cubic_surfaces_with_double_sixes_activity_description
      *Descr;
   surface_classify_wedge *SCW;

   classification_of_cubic_surfaces_with_double_sixes_activity();
   ~classification_of_cubic_surfaces_with_double_sixes_activity();
   void init(
         classification_of_cubic_surfaces_with_double_sixes_activity_description
            *Descr,
         surface_classify_wedge *SCW,
         int verbose_level);
   void perform_activity(int verbose_level);
   void report(
         poset_classification::poset_classification_report_options
```

```
            *report_options,
         int verbose_level);
   void do_surface_identify_Eckardt(int verbose_level);
   void do_surface_identify_F13(int verbose_level);
   void do_surface_identify_Bes(int verbose_level);
   void do_surface_identify_general_abcd(int verbose_level);
   void do_surface_isomorphism_testing(
         cubic_surfaces_in_general::surface_create_description
            *surface_descr_isomorph1,
         cubic_surfaces_in_general::surface_create_description
            *surface_descr_isomorph2,
         int verbose_level);
   void do_recognize(
         cubic_surfaces_in_general::surface_create_description
            *surface_descr,
         int verbose_level);
   void do_write_source_code(int verbose_level);
   void do_sweep_Cayley(
         int verbose_level);


};



// ##############################################################################
// classify_five_plus_one.cpp
// ##############################################################################

//! classification of five plus one sets of lines in PG(3,q). A five plus one is
five pairwise skew lines with a common transversal.


class classify_five_plus_one {

public:

   int q;
   field_theory::finite_field *F;
   actions::action *A;

   cubic_surfaces_in_general::surface_with_action *Surf_A;
   algebraic_geometry::surface_domain *Surf;


   actions::action *A2;
      // the action on the wedge product
```

```
induced_actions::action_on_wedge_product *AW;
   // internal data structure for the wedge action

int *Elt0; // used in identify_five_plus_one
int *Elt1; // used in identify_five_plus_one

groups::strong_generators *SG_line_stab;
   // stabilizer of the special line in PGL(4,q)
   // this group acts on the set Neighbors[] in the wedge action


orthogonal_geometry::linear_complex *Linear_complex;



ring_theory::longinteger_object go, stab_go;
groups::sims *Stab;
groups::strong_generators *stab_gens;

int *orbit;
int orbit_len;

long int pt0_idx_in_orbit;




actions::action *A_on_neighbors;
   // restricted action A2 on the set Neighbors[]

poset_classification::poset_classification_control *Control;
poset_classification::poset_with_group_action *Poset;
poset_classification::poset_classification *Five_plus_one;
   // orbits on five-plus-one configurations

int *Pts_for_partial_ovoid_test; // [5*6]


classify_five_plus_one();
~classify_five_plus_one();
void init(
      cubic_surfaces_in_general::surface_with_action
         *Surf_A,
      poset_classification::poset_classification_control
         *Control,
      int verbose_level);
void classify_partial_ovoids(
```

```
      int verbose_level);
   int line_to_neighbor(
         long int line_rk, int verbose_level);
   void partial_ovoid_test_early(
         long int *S, int len,
      long int *candidates, int nb_candidates,
      long int *good_candidates, int &nb_good_candidates,
      int verbose_level);
   void identify_five_plus_one(
         long int *five_lines,
         long int transversal_line,
      long int *five_lines_out_as_neighbors,
      int &orbit_index,
      int *transporter, int verbose_level);
   void report(
         std::ostream &ost,
         graphics::layered_graph_draw_options
            *draw_options,
         poset_classification::poset_classification_report_options
            *Opt,
         int verbose_level);


};


// ##############################################################################
// classify_double_sixes.cpp
// ##############################################################################

//! classification of double sixes in PG(3,q)


class classify_double_sixes {

public:

   classify_five_plus_one *Five_p1;

   int *Elt3; // used in upstep

   int len;
      // = gen->nb_orbits_at_level(5)
      // = number of orbits on 5-sets of lines
   int *Idx;
      // Idx[nb], list of orbits
      // for which the rank of the system is equal to 19
```

```
   int nb; // number of good orbits
   int *Po;
      // Po[Flag_orbits->nb_flag_orbits],
      //list of orbits for which a double six exists



   invariant_relations::flag_orbits *Flag_orbits;

   invariant_relations::classification_step *Double_sixes;


   classify_double_sixes();
   ~classify_double_sixes();
   void init(
         classify_five_plus_one *Five_p1,
         int verbose_level);
   void test_orbits(int verbose_level);
   void classify(int verbose_level);
   void downstep(int verbose_level);
   void upstep(int verbose_level);
   void print_five_plus_ones(std::ostream &ost);
   void identify_double_six(
         long int *double_six,
      int *transporter, int &orbit_index,
      int verbose_level);
   void make_spreadsheet_of_fiveplusone_configurations(
         data_structures::spreadsheet *&Sp,
      int verbose_level);
   void write_file(
         std::ofstream &fp, int verbose_level);
   void read_file(
         std::ifstream &fp, int verbose_level);
};



// ##############################################################################
// surface_classify_wedge.cpp
// ##############################################################################

//! classification of cubic surfaces using double sixes as substructures


class surface_classify_wedge {
public:
   field_theory::finite_field *F;
```

```
int q;

std::string fname_base;

actions::action *A; // the action of PGL(4,q) on points
actions::action *A2; // the action on the wedge product

algebraic_geometry::surface_domain *Surf;
cubic_surfaces_in_general::surface_with_action *Surf_A;

int *Elt0;
int *Elt1;
int *Elt2;
int *Elt3;

// substructures:

classify_five_plus_one *Five_p1;

classify_double_sixes *Classify_double_sixes;


// classification of cubic surfaces:

invariant_relations::flag_orbits *Flag_orbits;

invariant_relations::classification_step *Surfaces;


// created by post_process():

int nb_surfaces;
data_structures_groups::set_and_stabilizer *SaS;
   // [nb_surfaces]

long int *Lines; // [nb_surfaces * 27]
int *Eqn; // [nb_surfaces * 20]



surface_classify_wedge();
~surface_classify_wedge();
void init(
      cubic_surfaces_in_general::surface_with_action
         *Surf_A,
   poset_classification::poset_classification_control
      *Control,
```

```
   int verbose_level);
void do_classify_double_sixes(int verbose_level);
void do_classify_surfaces(int verbose_level);
void classify_surfaces_from_double_sixes(
      int verbose_level);
void post_process(int verbose_level);
void downstep(int verbose_level);
void upstep(int verbose_level);
void derived_arcs(int verbose_level);
void starter_configurations_which_are_involved(
      int iso_type,
   int *&Starter_configuration_idx,
   int &nb_starter_conf, int verbose_level);
void write_file(
      std::ofstream &fp, int verbose_level);
void read_file(
      std::ifstream &fp, int verbose_level);

void identify_Eckardt_and_print_table(int verbose_level);
void identify_F13_and_print_table(int verbose_level);
void identify_Bes_and_print_table(int verbose_level);
void identify_Eckardt(
      int *Iso_type, int *Nb_lines, int verbose_level);
void identify_F13(
   int *Iso_type, int *Nb_lines, int verbose_level);
void identify_Bes(
   int *Iso_type, int *Nb_lines, int verbose_level);
int isomorphism_test_pairwise(
      cubic_surfaces_in_general::surface_create *SC1,
      cubic_surfaces_in_general::surface_create *SC2,
   int &isomorphic_to1, int &isomorphic_to2,
   int *Elt_isomorphism_1to2,
   int verbose_level);
void identify_surface(int *coeff_of_given_surface,
   int &isomorphic_to, int *Elt_isomorphism,
   int verbose_level);
void latex_surfaces(
      std::ostream &ost,
      int f_with_stabilizers, int verbose_level);
void report_surface(
      std::ostream &ost,
      int orbit_index, int verbose_level);
void generate_source_code(int verbose_level);
void generate_history(int verbose_level);
int test_if_surfaces_have_been_computed_already();
void write_surfaces(int verbose_level);
void read_surfaces(int verbose_level);
```

```
   int test_if_double_sixes_have_been_computed_already();
   void write_double_sixes(int verbose_level);
   void read_double_sixes(int verbose_level);
   void create_report(
         int f_with_stabilizers,
         graphics::layered_graph_draw_options *draw_options,
         poset_classification::poset_classification_report_options *Opt,
         int verbose_level);
   void report(
         std::ostream &ost,
         int f_with_stabilizers,
         graphics::layered_graph_draw_options *draw_options,
         poset_classification::poset_classification_report_options *Opt,
         int verbose_level);
   void create_report_double_sixes(
         int verbose_level);
   void test_isomorphism(
         cubic_surfaces_in_general::surface_create_description
            *Descr1,
         cubic_surfaces_in_general::surface_create_description
            *Descr2,
         int verbose_level);
   void recognition(
         cubic_surfaces_in_general::surface_create_description
            *Descr,
         int verbose_level);
   void sweep_Cayley(int verbose_level);
   void identify_general_abcd(
      int *Iso_type, int *Nb_lines, int verbose_level);
   void identify_general_abcd_and_print_table(int verbose_level);

};


}}}}




#endif /* SRC_LIB_TOP_LEVEL_SURFACES_SURFACES_AND_DOUBLE_SIXES_SURFACES_AND_DOUBL
E_SIXES_H_ */
```

7.16 Cubic Surfaces in General

```
/*
 * surfaces_general.h
 *
 *  Created on: Jun 26, 2021
 *      Author: betten
 */

#ifndef SRC_LIB_TOP_LEVEL_SURFACES_SURFACES_GENERAL_SURFACES_GENERAL_H_
#define SRC_LIB_TOP_LEVEL_SURFACES_SURFACES_GENERAL_SURFACES_GENERAL_H_



namespace orbiter {
namespace layer5_applications {
namespace applications_in_algebraic_geometry {
namespace cubic_surfaces_in_general {


// ##############################################################################
// cubic_surface_activity_description.cpp
// ##############################################################################

//! description of an activity associated with a cubic surface


class cubic_surface_activity_description {
public:

   int f_report;

   int f_export_something;
   std::string export_something_what;

   int f_export_gap;

   int f_all_quartic_curves;

   int f_export_all_quartic_curves;


   cubic_surface_activity_description();
   ~cubic_surface_activity_description();
   int read_arguments(
      int argc, std::string *argv,
      int verbose_level);
   void print();
```

```
};

// ##############################################################################
// cubic_surface_activity.cpp
// ##############################################################################

//! an activity associated with a cubic surface


class cubic_surface_activity {
public:

   cubic_surface_activity_description *Descr;
   surface_create *SC;


   cubic_surface_activity();
   ~cubic_surface_activity();
   void init(
         cubic_surfaces_in_general::cubic_surface_activity_description
            *Cubic_surface_activity_description,
         surface_create *SC, int verbose_level);
   void perform_activity(int verbose_level);
};




// ##############################################################################
// surface_clebsch_map.cpp
// ##############################################################################

//! a Clebsch map associated with a cubic surface and a choice of half double six
, to be used in surface_create_by_arc_lifting::init


class surface_clebsch_map {
public:

   surface_object_with_action *SOA;

   int orbit_idx;
   int f, l, hds;

   algebraic_geometry::clebsch_map *Clebsch_map;
```

```
   surface_clebsch_map();
   ~surface_clebsch_map();
   void report(std::ostream &ost, int verbose_level);
   void init(surface_object_with_action *SOA,
         int orbit_idx, int verbose_level);

};


// ##############################################################################
// surface_create.cpp
// ##############################################################################


//! to create a cubic surface from a description using class surface_create_descr
iption


class surface_create {

public:
   surface_create_description *Descr;

   std::string prefix;
   std::string label_txt;
   std::string label_tex;

   int f_ownership;

   int q;
   field_theory::finite_field *F;

   int f_semilinear;

   projective_geometry::projective_space_with_action *PA;

   algebraic_geometry::surface_domain *Surf;

   surface_with_action *Surf_A;

   algebraic_geometry::surface_object *SO;

   int f_has_group;
   groups::strong_generators *Sg;
   int f_has_nice_gens;
   data_structures_groups::vector_ge *nice_gens;
```

```
surface_object_with_action *SOA;


surface_create();
~surface_create();
void create_cubic_surface(
      surface_create_description *Descr,
      int verbose_level);
int init_with_data(surface_create_description *Descr,
   surface_with_action *Surf_A,
   int verbose_level);
int init(surface_create_description *Descr,
   int verbose_level);
int create_surface_from_description(int verbose_level);
void override_group(std::string &group_order_text,
      int nb_gens, std::string &gens_text, int verbose_level);
void create_Eckardt_surface(int a, int b, int verbose_level);
void create_surface_G13(int a, int verbose_level);
void create_surface_F13(int a, int verbose_level);
void create_surface_bes(int a, int c, int verbose_level);
void create_surface_general_abcd(int a, int b, int c, int d,
      int verbose_level);
void create_surface_by_coefficients(std::string &coefficients_text,
      std::vector<std::string> &select_double_six_string,
      int verbose_level);
void create_surface_by_coefficient_vector(int *coeffs20,
      std::vector<std::string> &select_double_six_string,
      int verbose_level);
void create_surface_by_rank(std::string &rank_text, int defining_q,
      std::vector<std::string> &select_double_six_string,
      int verbose_level);
void create_surface_from_catalogue(int iso,
      std::vector<std::string> &select_double_six_string,
      int verbose_level);
void create_surface_by_arc_lifting(
      std::string &arc_lifting_text,
      int verbose_level);
void create_surface_by_arc_lifting_with_two_lines(
      std::string &arc_lifting_text,
      std::string &arc_lifting_two_lines_text,
      int verbose_level);
void create_surface_Cayley_form(
      int k, int l, int m, int n,
      int verbose_level);
int create_surface_by_equation(
      std::string &name_of_formula,
```

```
        std::string &name_of_formula_tex,
        std::string &managed_variables,
        std::string &equation_text,
        std::string &equation_parameters,
        std::string &equation_parameters_tex,
        std::vector<std::string> &select_double_six_string,
        int verbose_level);
   void create_surface_by_double_six(
        std::string &by_double_six_label,
        std::string &by_double_six_label_tex,
        std::string &by_double_six_text,
        int verbose_level);
   void create_surface_by_skew_hexagon(
        std::string &given_label,
        std::string &given_label_tex,
        int verbose_level);
   void apply_transformations(
      std::vector<std::string> &transform_coeffs,
      std::vector<int> &f_inverse_transform,
      int verbose_level);
   // applies all transformations and then recomputes the properties
   void apply_single_transformation(int f_inverse,
        int *transformation_coeffs,
        int sz,
        int verbose_level);
   // transforms SO->eqn, SO->Lines and SO->Pts,
   // Also transforms Sg (if f_has_group is TRUE)
#if 0
   void compute_group(
        projective_geometry::projective_space_with_action *PA,
        int verbose_level);
      // not working ToDo
#endif
   void export_something(std::string &what, int verbose_level);
   void export_gap(int verbose_level);
   void do_report(int verbose_level);
   void do_report2(std::ostream &ost, int verbose_level);
   void report_with_group(
        std::string &Control_six_arcs_label,
        int verbose_level);
   void test_group(int verbose_level);

};


// ##############################################################################
// surface_create_description.cpp
```

```
// ##############################################################################


//! to describe a cubic surface from the command line


class surface_create_description {

public:

   int f_space;
   std::string space_label;

   int f_space_pointer;
   projective_geometry::projective_space_with_action *space_pointer;


   int f_label_txt;
   std::string label_txt;

   int f_label_tex;
   std::string label_tex;

   int f_label_for_summary;
   std::string label_for_summary;

   int f_catalogue;
   int iso;
   int f_by_coefficients;
   std::string coefficients_text;

   int f_by_rank;
   std::string rank_text;
   int rank_defining_q;

   int f_family_Eckardt;
   int family_Eckardt_a;
   int family_Eckardt_b;

   int f_family_G13;
   int family_G13_a;

   int f_family_F13;
   int family_F13_a;

   int f_family_bes;
```

```
int family_bes_a;
int family_bes_c;

int f_family_general_abcd;
int family_general_abcd_a;
int family_general_abcd_b;
int family_general_abcd_c;
int family_general_abcd_d;

int f_arc_lifting;
std::string arc_lifting_text;
std::string arc_lifting_two_lines_text;

int f_arc_lifting_with_two_lines;
std::vector<std::string> select_double_six_string;

int f_Cayley_form;
int Cayley_form_k;
int Cayley_form_l;
int Cayley_form_m;
int Cayley_form_n;


int f_by_equation;
std::string equation_name_of_formula;
std::string equation_name_of_formula_tex;
std::string equation_managed_variables;
std::string equation_text;
std::string equation_parameters;
std::string equation_parameters_tex;


int f_by_double_six;
std::string by_double_six_label;
std::string by_double_six_label_tex;
std::string by_double_six_text;

int f_by_skew_hexagon;
std::string by_skew_hexagon_label;
std::string by_skew_hexagon_label_tex;

int f_override_group;
std::string override_group_order;
int override_group_nb_gens;
std::string override_group_gens;

std::vector<std::string> transform_coeffs;
```

```
   std::vector<int> f_inverse_transform;



   surface_create_description();
   ~surface_create_description();
   int read_arguments(int argc, std::string *argv,
      int verbose_level);
   void print();
};


// ##############################################################################
// surface_domain_high_level.cpp
// ##############################################################################


//! high level functions for cubic surfaces


class surface_domain_high_level {

public:


   surface_domain_high_level();
   ~surface_domain_high_level();

   void do_sweep_4_15_lines(
         projective_geometry::projective_space_with_action *PA,
         surface_create_description *Surface_Descr,
         std::string &sweep_fname,
         int verbose_level);
   void do_sweep_F_beta_9_lines(
         projective_geometry::projective_space_with_action *PA,
         surface_create_description *Surface_Descr,
         std::string &sweep_fname,
         int verbose_level);
   void do_sweep_6_9_lines(
         projective_geometry::projective_space_with_action *PA,
         surface_create_description *Surface_Descr,
         std::string &sweep_fname,
         int verbose_level);
   void do_sweep_4_27(
         projective_geometry::projective_space_with_action *PA,
         surface_create_description *Surface_Descr,
         std::string &sweep_fname,
```

```
        int verbose_level);
void do_sweep_4_L9_E4(
        projective_geometry::projective_space_with_action *PA,
        surface_create_description *Surface_Descr,
        std::string &sweep_fname,
        int verbose_level);


void classify_surfaces_with_double_sixes(
        projective_geometry::projective_space_with_action *PA,
        std::string &control_label,
        cubic_surfaces_and_double_sixes::surface_classify_wedge *&SCW,
        int verbose_level);


void do_study_surface(
        field_theory::finite_field *F,
        int nb, int verbose_level);
void do_classify_surfaces_through_arcs_and_two_lines(
        projective_geometry::projective_space_with_action *PA,
        std::string &Control_six_arcs_label,
        int f_test_nb_Eckardt_points, int nb_E,
        int verbose_level);
void do_classify_surfaces_through_arcs_and_trihedral_pairs(
        projective_geometry::projective_space_with_action *PA,
        poset_classification::poset_classification_control *Control1,
        poset_classification::poset_classification_control *Control2,
        std::string &Control_six_arcs_label,
        int f_test_nb_Eckardt_points, int nb_E,
        int verbose_level);
void do_six_arcs(
        projective_geometry::projective_space_with_action *PA,
        std::string &Control_six_arcs_label,
        int f_filter_by_nb_Eckardt_points, int nb_Eckardt_points,
        int verbose_level);
void do_cubic_surface_properties(
        projective_geometry::projective_space_with_action *PA,
        std::string &fname_csv, int defining_q,
        int column_offset,
        int verbose_level);
void do_cubic_surface_properties_analyze(
        projective_geometry::projective_space_with_action *PA,
        std::string &fname_csv, int defining_q,
        int verbose_level);
void report_singular_surfaces(
        std::ostream &ost,
        struct cubic_surface_data_set *Data,
```

```
        int nb_orbits, int verbose_level);
   void report_non_singular_surfaces(
        std::ostream &ost,
        struct cubic_surface_data_set *Data,
        int nb_orbits, int verbose_level);
   void report_surfaces_by_lines(
        std::ostream &ost,
        struct cubic_surface_data_set *Data,
        data_structures::tally &T, int verbose_level);
   void do_create_surface_reports(
        std::string &field_orders_text, int verbose_level);
   void do_create_surface_atlas(
        int q_max, int verbose_level);
   void do_create_surface_atlas_q_e(
        int q_max,
        struct table_surfaces_field_order *T,
        int nb_e, int *Idx, int nb,
        std::string &fname_report_tex,
        int verbose_level);
   void do_create_dickson_atlas(int verbose_level);
   void make_fname_surface_report_tex(
        std::string &fname, int q, int ocn);
   void make_fname_surface_report_pdf(
        std::string &fname, int q, int ocn);

};


// ##############################################################################
// surface_object_with_action.cpp
// ##############################################################################


//! an instance of a cubic surface together with its stabilizer


class surface_object_with_action {

public:

   int q;
   field_theory::finite_field *F; // do not free

   algebraic_geometry::surface_domain *Surf; // do not free
   surface_with_action *Surf_A; // do not free

   algebraic_geometry::surface_object *SO; // do not free
```

```
groups::strong_generators *Aut_gens;
   // generators for the automorphism group

int f_has_nice_gens;
data_structures_groups::vector_ge *nice_gens;

groups::strong_generators *projectivity_group_gens;
groups::sylow_structure *Syl;

actions::action *A_on_points;
actions::action *A_on_Eckardt_points;
actions::action *A_on_Double_points;
actions::action *A_on_Single_points;
actions::action *A_on_the_lines;
actions::action *A_single_sixes;
actions::action *A_double_sixes;
actions::action *A_on_tritangent_planes;
actions::action *A_on_Hesse_planes;
actions::action *A_on_axes;
actions::action *A_on_trihedral_pairs;
actions::action *A_on_pts_not_on_lines;


groups::schreier *Orbits_on_points;
groups::schreier *Orbits_on_Eckardt_points;
groups::schreier *Orbits_on_Double_points;
groups::schreier *Orbits_on_Single_points;
groups::schreier *Orbits_on_lines;
groups::schreier *Orbits_on_single_sixes;
groups::schreier *Orbits_on_double_sixes;
groups::schreier *Orbits_on_tritangent_planes;
groups::schreier *Orbits_on_Hesse_planes;
groups::schreier *Orbits_on_axes;
groups::schreier *Orbits_on_trihedral_pairs;
groups::schreier *Orbits_on_points_not_on_lines;



surface_object_with_action();
~surface_object_with_action();
void init_equation(surface_with_action *Surf_A, int *eqn,
      groups::strong_generators *Aut_gens, int verbose_level);
void init_with_group(surface_with_action *Surf_A,
   long int *Lines, int nb_lines, int *eqn,
   groups::strong_generators *Aut_gens,
   int f_find_double_six_and_rearrange_lines,
   int f_has_nice_gens,
```

```
   data_structures_groups::vector_ge *nice_gens,
   int verbose_level);
void init_with_surface_object(surface_with_action *Surf_A,
      algebraic_geometry::surface_object *SO,
      groups::strong_generators *Aut_gens,
      int f_has_nice_gens,
      data_structures_groups::vector_ge *nice_gens,
      int verbose_level);
void init_surface_object(surface_with_action *Surf_A,
      algebraic_geometry::surface_object *SO,
      groups::strong_generators *Aut_gens, int verbose_level);
void compute_projectivity_group(int verbose_level);
void compute_orbits_of_automorphism_group(int verbose_level);
void init_orbits_on_points(int verbose_level);
void init_orbits_on_Eckardt_points(int verbose_level);
void init_orbits_on_Double_points(int verbose_level);
void init_orbits_on_Single_points(int verbose_level);
void init_orbits_on_lines(int verbose_level);
void init_orbits_on_half_double_sixes(int verbose_level);
void init_orbits_on_double_sixes(int verbose_level);
void init_orbits_on_tritangent_planes(int verbose_level);
void init_orbits_on_Hesse_planes(int verbose_level);
void init_orbits_on_axes(int verbose_level);
void init_orbits_on_trihedral_pairs(int verbose_level);
void init_orbits_on_points_not_on_lines(int verbose_level);
void print_generators_on_lines(
      std::ostream &ost,
      groups::strong_generators *Aut_gens,
      int verbose_level);
void print_elements_on_lines(
      std::ostream &ost,
      groups::strong_generators *Aut_gens,
      int verbose_level);
void print_automorphism_group(std::ostream &ost,
   int f_print_orbits, std::string &fname_mask,
   graphics::layered_graph_draw_options *Opt,
   int verbose_level);
void cheat_sheet_basic(std::ostream &ost, int verbose_level);
void cheat_sheet(std::ostream &ost,
      std::string &label_txt,
      std::string &label_tex,
      int f_print_orbits, std::string &fname_mask,
      graphics::layered_graph_draw_options *Opt,
      int verbose_level);
void print_automorphism_group_generators(
      std::ostream &ost, int verbose_level);
void investigate_surface_and_write_report(
```

```
        graphics::layered_graph_draw_options *Opt,
        actions::action *A,
        surface_create *SC,
        cubic_surfaces_and_arcs::six_arcs_not_on_a_conic *Six_arcs,
        int verbose_level);
   void investigate_surface_and_write_report2(
        std::ostream &ost,
        graphics::layered_graph_draw_options *Opt,
        actions::action *A,
        surface_create *SC,
        cubic_surfaces_and_arcs::six_arcs_not_on_a_conic *Six_arcs,
        std::string &fname_mask,
        std::string &label,
        std::string &label_tex,
        int verbose_level);
   void all_quartic_curves(
        std::string &surface_label_txt,
        std::string &surface_label_tex,
        std::ostream &ost,
        int verbose_level);
   void export_all_quartic_curves(
        std::ostream &ost_quartics_csv,
        int verbose_level);
   void print_full_del_Pezzo(std::ostream &ost, int verbose_level);
   void print_everything(std::ostream &ost, int verbose_level);
   void print_summary(std::ostream &ost);
   void print_action_on_surface(
        std::string &label_of_elements,
        int *element_data, int nb_elements,
        int verbose_level);
   void print_double_sixes(std::ostream &ost);

};

// ##############################################################################
// surface_study.cpp
// ##############################################################################

//! to study properties of cubic surfaces

class surface_study {
public:
   int q;
   int nb;
   int *rep;
   std::string prefix;
   field_theory::finite_field *F;
```

```
algebraic_geometry::surface_domain *Surf;

int nb_lines_PG_3;

int *data;
int nb_gens;
int data_size;
std::string stab_order;

actions::action *A;
actions::action *A2;
groups::sims *S;
long int *Lines;
int *coeff;

int f_semilinear;

data_structures_groups::set_and_stabilizer *SaS;


// line orbits:
int *orbit_first;
int *orbit_length;
int *orbit;
int nb_orbits;


// orbit_on_lines:
actions::action *A_on_lines;
groups::schreier *Orb;
int shortest_line_orbit_idx;

// for study_find_eckardt_points:
int *Adj;
int *R;
long int *Intersection_pt;
long int *Double_pts;
int nb_double_pts;
long int *Eckardt_pts;
int nb_Eckardt_pts;


void init(field_theory::finite_field *F, int nb, int verbose_level);
void study_intersection_points(int verbose_level);
void study_line_orbits(int verbose_level);
void study_group(int verbose_level);
void study_orbits_on_lines(int verbose_level);
```

```
   void study_find_eckardt_points(int verbose_level);
   void study_surface_with_6_eckardt_points(int verbose_level);
};




// ##############################################################################
// surface_with_action.cpp
// ##############################################################################

//! cubic surfaces in projective space with automorphism group



class surface_with_action {

public:


   projective_geometry::projective_space_with_action *PA;

   int f_semilinear;

   algebraic_geometry::surface_domain *Surf; // do not free

   actions::action *A; // linear group PGGL(4,q)

   actions::action *A_wedge; // linear group PGGL(4,q)


   actions::action *A2; // linear group PGGL(4,q) acting on lines
   actions::action *A_on_planes; // linear group PGGL(4,q) acting on planes

   int *Elt1;

   induced_actions::action_on_homogeneous_polynomials *AonHPD_3_4;


   cubic_surfaces_and_arcs::classify_trihedral_pairs
      *Classify_trihedral_pairs;

   geometry::spread_domain *SD;
   spreads::recoordinatize *Recoordinatize;
   long int *regulus; // [regulus_size]
   int regulus_size; // q + 1
```

```
surface_with_action();
~surface_with_action();
void init(
      algebraic_geometry::surface_domain *Surf,
      projective_geometry::projective_space_with_action *PA,
      int f_recoordinatize,
      int verbose_level);
void complete_skew_hexagon(
   long int *skew_hexagon,
   std::vector<std::vector<long int> > &Double_sixes,
   int verbose_level);
void complete_skew_hexagon_with_polarity(
      std::string &label_for_printing,
   long int *skew_hexagon,
   int *Polarity36,
   std::vector<std::vector<long int> > &Double_sixes,
   int verbose_level);
void create_regulus_and_opposite_regulus(
   long int *three_skew_lines, long int *&regulus,
   long int *&opp_regulus, int &regulus_sz,
   int verbose_level);
int create_double_six_safely(
   long int *five_lines, long int transversal_line,
   long int *double_six, int verbose_level);
int create_double_six_from_five_lines_with_a_common_transversal(
   long int *five_lines, long int transversal_line,
   long int *double_six, int verbose_level);
void report_basics(std::ostream &ost);
void report_double_triplets(std::ostream &ost);
void report_double_triplets_detailed(std::ostream &ost);
void sweep_4_15_lines(
      surface_create_description *Surface_Descr,
      std::string &sweep_fname,
      int verbose_level);
void sweep_F_beta_9_lines(
      surface_create_description *Surface_Descr,
      std::string &sweep_fname,
      int verbose_level);
void sweep_6_9_lines(
      surface_create_description *Surface_Descr,
      std::string &sweep_fname,
      int verbose_level);
void sweep_4_27(
      surface_create_description *Surface_Descr,
      std::string &sweep_fname,
```

```
        int verbose_level);
   void sweep_4_L9_E4(
        surface_create_description *Surface_Descr,
        std::string &sweep_fname,
        int verbose_level);
   void table_of_cubic_surfaces(
        int verbose_level);
   void table_of_cubic_surfaces_export_csv(
        long int *Table,
        int nb_cols,
        int q, int nb_cubic_surfaces,
        surface_create **SC,
        int verbose_level);
   void table_of_cubic_surfaces_export_sql(
        long int *Table,
        int nb_cols,
        int q, int nb_cubic_surfaces,
        surface_create **SC,
        int verbose_level);

};



}}}}



#endif /* SRC_LIB_TOP_LEVEL_SURFACES_SURFACES_GENERAL_SURFACES_GENERAL_H_ */
```