GAP4
IV. Groups and Homomorphisms

Alexander Hulpke

June 1, 1998

`http://www-gap.dcs.st-and.ac.uk/~ahulpke/course.html`

Abstract

This weeks lecture is a bit "in between". It explains concepts about groups and homomorphisms that are not difficult, but which we will need later again and again.

Groups

Usually groups are generated as `Group` or `Subgroup` from generators. These generators are stored in the attribute `GeneratorsOfGroup`.

There are a couple of generic operations to construct parametrized groups:

```
SymmetricGroup(5);
DihedralGroup(IsPermGroup,10);
GL(5,4);
```

(Methods for the operations are handled slightly differently to cope with the filter as first argument.)

There also are extensive libraries of groups. Selection functions permit to fetch only those groups which have certain properties:

```
AllTransitiveGroups(DegreeOperation,12,Size,1440,
    i->Length(DerivedSeries(i)),4);
```

Groups are collections of objects that permit operations like `Size` or `in`. Their family is the `CollectionsFamily` of their elements. `AsList` and `Enumerator` yield a list of elements.

There are many special operations for groups like `RightTransversal` or `Normalizer`. Some of these store the result wrt. the `Parent` and call operations like `NormalizerOp`.

Subgroup series like `ElementaryAbelianSeries`, `LowerCentralSeries` are given as lists of subgroups in decreasing size.

`ClosureGroup` extends a subgroup by element(s), `ConjugateSubgroup` produces a conjugate image.

In some cases (like `IsSolvableGroup` or `DerivedSeriesOfGroup`) when concepts are defined differently for other algebraic structures, an explicit operation for groups must be called.

Many operations (like conjugacy classes) are described best in the context of group operations which will be next week's topic.

Common types of groups

Permutation groups are created from permutations given in cycle form. Special operations for them include `MovedPoints` and `StabChain`.

GAP includes nearly-linear time Monte-Carlo methods for computing stabilizer chains. Performance can be improved substantially if the size is given.

Finitely presented groups are created as quotients of free groups. Most iof their methods use coset tables.

Pc groups are groups given by a polycyclic presentation. They are created from a rewriting system (which is a rather complicated process) or by converting a finite solvable group in this presentation. Quotient algorithms naturally create polycyclic presentations.

The generators of matrix groups are lists of vectors. Most methods for finite matrix groups work via a faithful permutation representation (*Nice monomorphism*, see later.)

Factor groups are created as image of the operation `NaturalHomomorphismByNormalSubgroup`, which will try to find an effective representation for the factor.

Mappings and Homomorphisms

General mappings are multivalued, the most important class are `IsSingleValued` mappings.

Mappings are equal if they have the same source and range and if they map all elements in the same way.

The family of a mapping is parametrized by the family of the source and the family of the range, this permits to check applicability by family predicates.

Mappings can be composed using `*` or `CompositionMapping`.

The basic user operations are `Source`, `Range`, `ImagesRepresentative`, `Image`, `PreImagesRepresentative` and `PreImage`.

Mapping properties include `IsInjective`, `IsSurjective`. Homomorphism properties are indicated by

`RespectsMultiplication`, `RespectsOne`, `RespectsInverses` (together `IsGroupGeneralMapping`), `RespectsAddition` and others.

Kernels are called accordingly:

`KernelOfMultiplicativeGeneralMapping` &c.

`GroupGeneralMappingByImages` will create a mapping from mapping group generators, `IsSingleValued` can be used to test homomorphism property.

In most cases, however `GroupHomomorphismByImages` or `GroupHomomorphismByImagesNC` (which will not check that it is indeed a homomorphism) should be called.

There is an attribute `AsGroupGeneralMappingByImages` that can be used to enforce an equal mapping in this representation.

It is possible to create groups from automorphisms (for example automorphisms of other groups). The operation `AutomorphismGroup` will compute the full automorphism group of a finite group.

An epimorphism from the free group can be used to express group elements as words in generators.

Changing a groups Representation

Homomorphisms are also used to obtain isomorphic groups in a different kind of representation. Operations are

- `IsomorphismPermGroup` which finds a faithful permutation representation (that might however be regular).
- `IsomorphismPcGroup` computes for finite solvable groups a pc system and creates the group.
- `IsomorphismFpGroup` choses generators in which a "short" presentation can be given.
- `IsomorphismFpGroupByGenerators` creates a presentation in given generators.

All these operations return an isomorphism from the old representation to the new. `Image` then obtains the isomorphic group.

Nice Monomorphisms

In some cases (matrix groups, automorphism groups) the best known methods are to translate the calculations in an isomorphic permutation group. The mechanism offered by GAP for this are *Nice monomorphisms*.

If the filter `IsHandledByNiceMonomorphism` is set, methods of high value become applicable that translate calculations in the `NiceObject` of the group, which is the image under the `NiceMonomorphism`.

The `NiceMonomorphism` must be applicable to group elements, therefore it often is an operation homomorphism.

The operation `AutomorphismGroup` will compute the full automorphism group of a finite group and automatically set a `NiceMonomorphism` based on faithful action on a characteristic subset. This will not happen automatically for groups generated by automorphisms.

The image of a nice monomorphism may depend on group generators or other parameters. The property `IsCanonicalNiceMonomorphism` is set if it depends only on the source as a set.

Group Products

GAP understands producrt constructions `DirectProduct`, `SemidirectProduct`, `SubdirectProduct`, `WreathProduct`.

For permutation groups the wreath product by standard is in its natural imprimitive action. There also is a command `WreathProductProductAction` that creates the (often primitive) product action.

At the moment methods exist only for permutation groups and pc groups.

A product `P` usually defines `Embedding(P,`*i*`)` and `Projection(P,`*i*`)` which are homomorphisms.

There is a more general mechanism for constructing extensions of Pc groups via cohomology.

Info and Assert

In many cases one wants an algorithm to output debugging information or to test internal sanity conditions. Both however should not be compulsory.

Information will be displayed according to the print level of the various info classes.

The command `NewInfoClass` creates an info class, `SetInfoLevel` assigns another print level (default 0).

The info classes defined in the library are called `Info`*Something*, for exaple `InfoPermGroup` and `InfoWarning`.

A call to `Info(`*class*`,`*level*`,`*obj1, obj2, . . .*`)` will print the objects if the level of the respective class is high enough. Otherwise the expressions for the objects are not evaluated.

Assertions work similarly: `SetAssertionLevel` assigns an assertion level (default 0). A call to `Assert(`*level*`,`*expr*`)` issues an error if *expr* does not evaluate to `true`. If the level is not high enough, the command is ignored.