An Overview of Computational Group Theory

(unter besonderer Berücksichtigung der Gruppenoperationen)

Alexander Hulpke
Department of Mathematics
Colorado State University
Fort Collins, CO, 80523, USA
http://www.hulpke.com

Ground Rules

Putting Groups on the Computer

To work with groups, we need to be able to work with their elements. Two ways have been established:

- Words in Generators and Relations (Finitely Presented).
  E.g. $\langle a, b \mid a^2 = b^3 = 1, ab = b^2a \rangle$. This is computationally hard (in general provably unsolvable!)
- Symmetries of objects. This is the natural setting for group actions.

Representing Symmetries

Symmetries are functions. Assuming we have some way to represent the underlying domain, we can represent functions as

- Permutations of a (finite) domain
- Matrices, representing the action on a vector space
- Functions (in the Computer Science sense), e.g. representing automorphisms.

Permutations are often possible, but can result in large degree

What To Store ?

Say we are willing to devote 2GB of memory.

$$\frac{2\mathrm{GB}}{100\mathrm{bytes}} = 21.5 \cdot 10^6$$

Each group element takes 100 bytes (permutation of degree 100, compressed 7×7 matrix over $\mathbb{F}_{13}$) — conservative estimate!

We cannot store all group elements unless we limit group orders severely!

What To Store !

In general we will store only generators of a group and will try to do with storing a limited number of elements (still could be many).

- Number of generators is $\log |G|$. In practice often ≤ 5 .
- Can use the group acting on itself to store only up to conjugacy.

Sometimes, finding generators for a symmetry group is already a stabilizer task in a larger group.

Fundamental Tasks

In this setup, we want to do:

ORDER find the order of a group. (Implies element test.)

HOMOM decompose an element as product in generators. (Homomorphisms!)

STAB determine the stabilizer of an object

TRANS find a group element mapping *a* to *b*

STRUCT determine the structure of a group (Holy Grail: Up to isomorphism)

Tools at Hand

- Arithmetic and Operations for underlying domains
- Linear Algebra
- Combinatorial (Exhaustive) searchPseudorandom Elements: Product Replacement (List of elements, replace x with x*y or x/y where x,y random list elements. Repeat).
- Group action: Orbits and Stabilizers

Part I
Orbits, Stabilizers and Stabilizer Chains

Basic Orbit Algorithm

The orbit ω^G of a point ω consists of all images ω^g for $g \in G$.

Each g itself is a product of generators (and inverses – assume included)

Thus it is sufficient to take iteratively images of all points obtained under all generators.

Input: $G = \langle g_1,\ldots,g_m \rangle$, acting on Ω. Seed $\omega \in \Omega$.
Output: The orbit ω^G.

```
begin
  Δ := [ω];
  for δ ∈ Δ do
    for i ∈ {1,...,m} do
      γ := δ^(g_i);
      if γ ∉ Δ then
        Append γ to Δ;
      fi;
    od;
  od;
  return Δ;
```

Modification: Transporters

For little extra cost we can also get transporter elements $T[\delta]$, such that $\omega^{T[\delta]} = \delta$. (A *Transversal*)

Note that these $T[\delta]$ are also representatives for the (right) cosets of $\mathrm{Stab}_G(\omega)$ in G.

If $\omega^g = \delta$ and $\omega^h = \gamma$, then $\delta^x = \gamma$ for $x = g^{-1}h$,

```
begin
  Δ := [ω]; T:=[1];
  for δ ∈ Δ do
    for i ∈ {1,...,m} do
      γ := δ^{g_i};
      if γ ∉ Δ then
        Append γ to Δ;
        Append T[δ] · g_i to T;
      fi;
    od;
  od;
  return Δ;
end.
```

Modification: Stabilizer

If $\omega^a = \delta$, $\delta^b = \gamma$, and $\omega^c = \gamma$, then $a \cdot b/c \in \mathrm{Stab}_G(\omega)$.

Schreier's lemma states that these elements (for all δ and all generators b) generate $\mathrm{Stab}_G(\omega)$.

```
begin
  Δ := [ω];T:=[1];S:=⟨1⟩;
  for δ ∈ Δ do
    for i ∈ {1,...,m} do
      γ := δ^{g_i};
      if γ ∉ Δ then
        Append γ to Δ;
        Append T[δ] · g_i to T;
      else
        S:=⟨S, T[δ] · g_i /T[γ]⟩;
      fi;
    od;
  od;
  return Δ;
end.
```

Remarks on the Algorithm

- Need to store whole orbit – memory limits feasibility.
- Store transversal T in factored form – *Schreier vector*. (Issue: balanced depth)
- Cost of basic algorithm is dominated by test $\gamma \notin \Delta$ to check for new points – use good data structures, e.g. hashing.
- There is a **huge** number of Schreier generators (one cannot reduce in theory). Need redundancy tests or random selection.

What Can We Do?

If the orbit algorithm is feasible, i.e. the orbit length not too long, we can thus solve

- TRANS, STAB, Action homomorphisms.
- ORDER,HOMOM (when acting by right multiplication) if group is not large
- Find all conjugates of an object, determine cosets of a subgroup (if element test) (and thus describe an orbit from within G)

To deal with larger cases, we need to use more group theory!

Variants

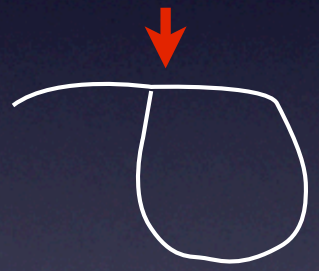
If storage requirements are beyond system capabilities the following variants can help:

- Use of Birthday paradox to estimate orbit length – indicate that stabilizer is known.
- Fuse orbits of subgroup $U<G$, e.g. by forming canonical U-representatives. Stabilizer is complicated unless U is normal.
- Is we can calculate subgroup orders, can stop if the largest proper divisor of [G:S] is smaller than the orbit length.

Tadpoles

To avoid finding canonical *U*-representatives:

Algorithmically assign (e.g. hash) to every $\omega \in \Omega$ an element g_ω. When γ is obtained as orbit element, calculate repeatedly γ^{g_γ}, until cycle (tadpole) is found
(cf. Pollard's ρ-method).
Store *only* cycle base.



Thus need to store one element per tadpole - save storage by a factor of tadpole size.

Divide and Conquer

Concentrate for the moment on ORDER, the action by right multiplication. The orbit is the set of all group elements.

But if $S \leq G$, we can factor the set of elements into S, and coset representatives for S in G.

Recursion to S represents G as cartesian product of transversals – logarithmic in $|G|$

If S is a stabilizer (for **any** action), we can find coset representatives via an transversal, and generators as Schreier generators.

Stabilizer Chains

Groups given as symmetries have a natural action on the underlying domain. We can pick some point ω, and use $S = \mathrm{Stab}_G(\omega)$.

When working with S, we pick another point to get again a subgroup as stabilizer. Eventually we get as points a **Base** $B = \{\beta_1,\ldots,\beta_m\} \subseteq \Omega$, such that no nontrivial element of G fixes all of B as well as a chain of subgroups

$$G > \mathrm{Stab}_G(\beta_1) > \mathrm{Stab}_G(\beta_1, \beta_2) > \cdots > \mathrm{Stab}_G(\beta_1,\ldots, \beta_m) = \langle 1 \rangle$$

For each subgroup, we also have a transversal

Base Images

As the tuple $(\beta_1,\ldots,\beta_m)$ is fixed only by the identity, every element $g\in G$ corresponds to a unique *base image* $(\beta_1^g,\ldots,\beta_m^g)$.

The transversal of $\mathrm{Stab}_G(\beta_1)$ in G contains an element x_1, so that $\beta_1^g=\beta_1^{x_1}$. Thus $g/x\in\mathrm{Stab}_G(\beta_1)$.

Iteratively, we get transversal elements x_i for $\mathrm{Stab}_G(\beta_1,\ldots,\beta_i)$ in $\mathrm{Stab}_G(\beta_1,\ldots,\beta_{i-1})$ so that the quotient $g/x_1/x_2/x_3/\cdots/x_m$ fixes all base points, i.e. $=1$ if $g\in G$.

This provides an element test for G.

Strong Generators

The transversal elements in turn are products of the stabilizer generators on the respective level.

The union of all stabilizer generators is called the set of *strong generators*.

Knowledge of the strong generators (for a given base) lets us rebuild the stabilizer chain easily.

The Schreier-Sims algorithm

To determine a stabilizer chain (or strong generators) we use a recursive process:

Assuming we already have a partial chain

- Add a further generator and extend the orbit.
- For each new Schreier generator, test (using the chain for the stabilizer) whether it is already in the stabilizer.
- Otherwise, add it as further generator to the stabilizer.

Consequences

Once we have a stabilizer chain, we can solve:

ORDER: The product of the orbit lengths is the product of the transversal lengths is $|G|$.

HOMOM: The element test decomposes elements as products of strong generators. Remember how these were obtained as products of original generators.

STABILIZER for points or tuples (by choosing a suitable base)

Subgroup Series (derived, etc.) and tests for solvability, nilpotentce, etc.

What actions?

For good performance, stabilizer indices (=orbit lengths) must be short. In the original case of S_n this is the case (though there is still a choice of base points).

In other situations (e.g. matrix groups, automorphism groups) it can be worth to use different actions (e.g. projective action, action on subgroups); there also is no good strategy for selecting base points.

But: Some groups (e.g. linear groups) do not have *any* subgroups of small index.

Performance Issues

Even if short orbits are available, the fundamental bottleneck is the number of Schreier generators.

Most are redundant, but testing takes time.

Overall runtime is polynomial (n^6), we would like (say $n=10^6$) nearly linear ($n \log(n) \log^c(n)$), which seems optimal (roughly $\log(n)$ generators of degree n).

Randomization

To eliminate redundant generators take

- Random Subset or
- Random Subproducts $\prod_{i=1}^{m} a_i^{\epsilon_i}$ where $\epsilon_i \in \{0, 1\}$ (better: assume m=10, only a_5 is new. Then any product with $\epsilon_5 = 1$ suffices, so we expect 2 subproducts versus >5 random elements)

of Schreier Generators, testing just a small number of generators in each level.

The resulting algorithm is **much** faster, but the result might be wrong.

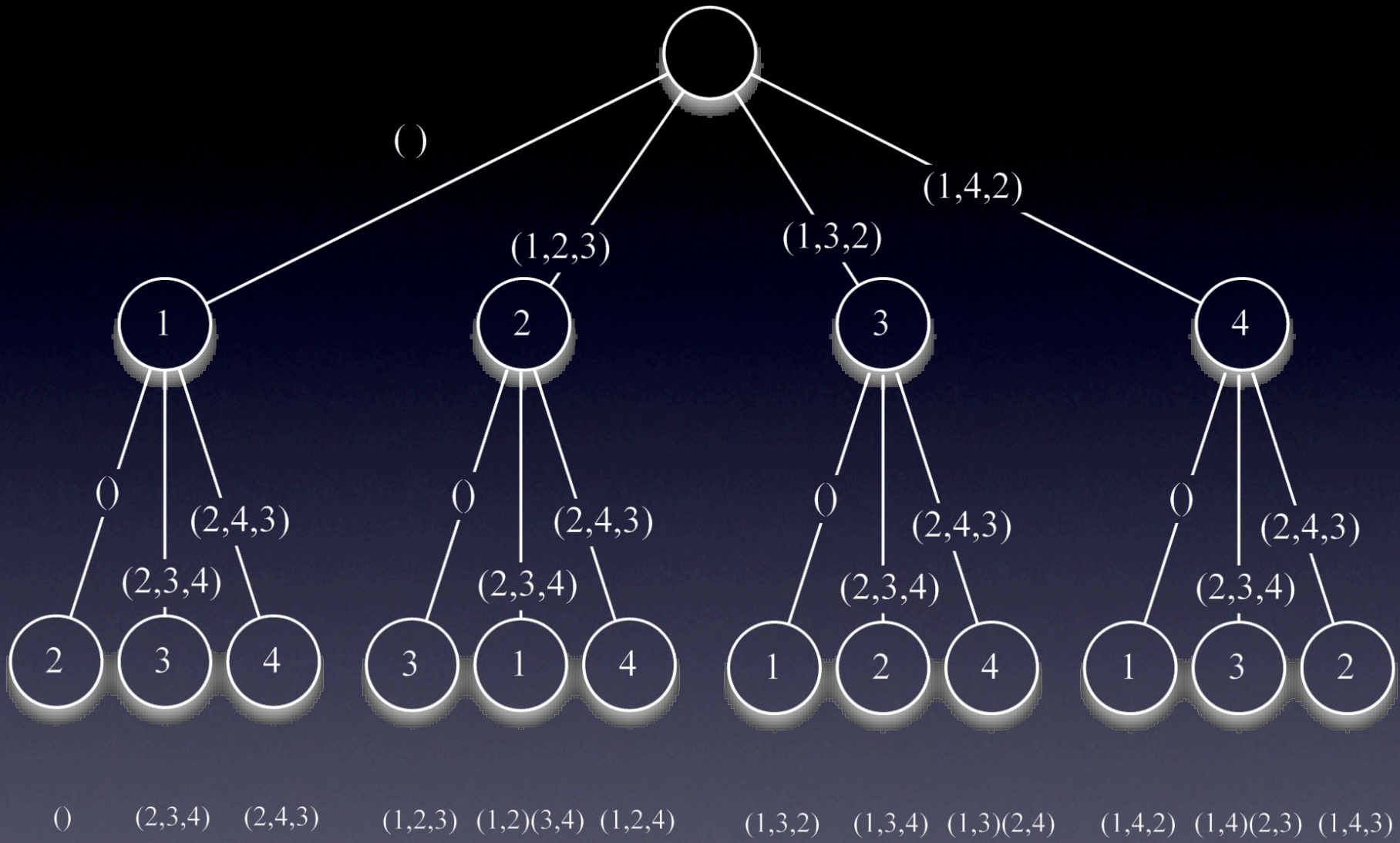
Verification

To verify correctness of a stabilizer chain, observe that a wrong chain will be *missing* parts. We can test using

- Combinatorial Test (comparatively expensive, no good description).
- Compare $|G|$ to the order claimed by the chain.
- Compute a presentation for G from the chain (see later) and evaluate it in the generators. If the chain is wrong, this will fail.

The resulting algorithm is *Las Vegas* – randomized good runtime + verification step.

Tree Structure



() (2,3,4) (2,4,3) (1,2,3) (1,2)(3,4) (1,2,4) (1,3,2) (1,3,4) (1,3)(2,4) (1,4,2) (1,4)(2,3) (1,4,3)

We can visualize the decomposition of G into a product of coset representatives as a tree.

Backtrack

One can traverse this tree in a depth-first search to test all elements of G for a particular property.

The process of doing this is called *Backtrack Search*.

It is used for stabilizer computations (or orbit equality by finding transporter elements) in cases where the ordinary orbit algorithm cannot succeed.

Pruning

To make this search feasible, it is necessary to prune tree branches. There are two sources for pruning:

- Problem-specific properties (together with use of appropriate base). E.g. when stabilizing $\{1,2\}$, if $1^x=2$, then $2^x=1$.
- Group-theoretic properties: The elements searched for typically lie in a double coset AxB. Try to test only the smallest element in each double coset. (Stand-in: Smallest in Ax and in xB, A and B computed "on the go").

Partition Backtrack

When descending in the tree, the combination of tree restrictions and problem restricttions can yield new pruning restrictions.

E.g. if we want to fix the set {1,2}, and we fix 1, we also need to fix 2.

Partition Backtrack (McKay, `nauty`) is a convenient way to process this: Every node corresponds to a partition of Ω. Descent in the tree corresponds to *refinement* (intersection) with the partition (i|1,2...i-1,i+1,...n). Problem-specific pruning similarly intersects with a partition (e.g. (1,2|3..n) for fixing {1,2}.)

Remarks

Backtrack is surprisingly fast if well implemented, but very clearly has limits.

The runtime for stabilizer calculations (transporter, or “in-orbit” calculations behave similar as the corresponding stabilizer) is related to the stabilizer index. Changing to a smaller group helps.

(Again, remember that some in some groups (symmetric, GL) almost all subgroups have huge index!)

Speeding up Backtrack

The key to speeding up backtrack (apart from good problem-specific pruning criteria) is to use intermediate subgroups.

Instead of solving the problem directly, try to find invariants of the object first and (if one can act on them) stabilize (or map) them first, descending to a subgroup.

If the invariants are a block system (system of imprimitivity) this is often done implicitly by replacing S_n by $S_a \wr S_b$. GL has similar subgroups.

BREAK

Part II
Using the Group Structure

Why Group Structure?

Knowledge of the group structure can help with group actions:

- Each transitive action is equivalent to action on cosets
- Find (maximal) subgroups that contain prospective stabilizers
- Homomorphisms provide approximations / Modification of orbit algorithm if there are normal subgroups
- Presentations for verifying a stabilizer chain

Composition Series

The main too towards structural computation is a composition series, i.e. a series

$$G = G_0 \rhd G_1 \rhd \cdots \rhd G_m = \langle 1 \rangle$$

such that G_i/G_{i+1} is simple. Jordan-Hölder: Uniqueness of factors with multiplicities.

To find such a series we just need to repeat the following step:

- Find a homomorphism with nontrivial kernel, recurse to image and kernel or
- Show that the group is simple

Homomorphisms

We will try to find homomorphisms φ as suitable actions. By keeping track of how an orbit is build, we get permutations for the image G^φ of the action homomorphism.

We process this image G^φ recursively and get a presentation of the image group.

Evaluating the relators for G^φ in pre-images of the generators in G yields (normal subgroup) generators for the kernel of φ.

Permutation groups

For permutation groups, the obvious actions are on orbits, or - if transitive - on blocks.
If G is transitive and has no blocks it's *primitive.*
The O'NAN-SCOTT theorem describes structure:

- Socle of G $\quad \langle N \mid N \lhd G \text{ minimal} \rangle$ has form $T \times \cdots \times T \lhd G$ with T simple.
- T abelian: affine group
- Otherwise action on socle is faithful. For each type of T get an image of the form $T^m \leq X \leq \mathrm{Aut}(T) \wr S_m$ with $[X:T^m]$ small.
- In base case $m=1$ work in G/T ad-hoc, or have proof of simplicity.

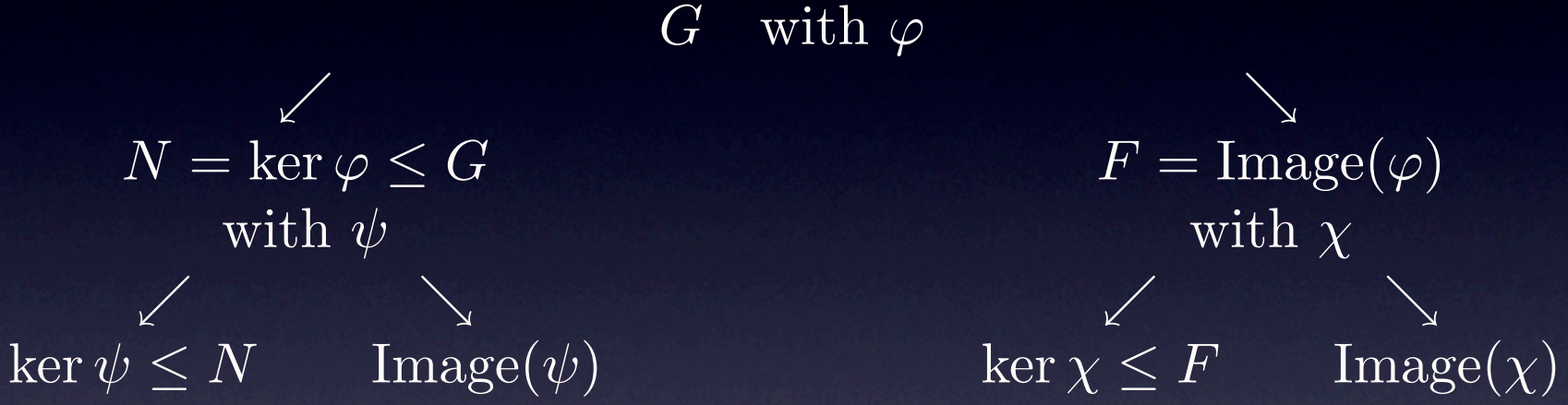
Matrix Groups

The Matrix group Recognition Project produces a similar setup for Matrix groups by considering the natural module:

- The initial reduction is reduction to submodules and factor modules. (MeatAxe)
- A second analog of intransitivity is tensor product decomposition. Imprimitivity exists for direct sum and tensor decompositions.
- ASCHBACHER's theorem provides a (more complicated) analog to O'NAN-SCOTT

Composition Tree

The resulting structure is called a *composition tree*:



At each node, we can evaluate the homomorphism (by acting on the objects of the underlying decomposition) and have generators for the kernel.

Presentations

If $N \triangleleft G$, we can combine presentations for N and for G/N.

- Generators for G/N become representatives in G
- Relators for G/N now evaluate in N.
- Added conjugation relators n^g=word in N.

For the base case use that presentations for all simple factors are known. (Efficiency: Short presentations, known for all groups but ${}^2G_2(q)$).

Modifying the series

By acting on the nonabelian composition factors (need: conjugacy tests in simple groups) we can obtain a new series

$$G \rhd R = R_0 \rhd R_1 \rhd \cdots \rhd R_k = \langle 1 \rangle$$

where

- R is the largest solvable normal subgroup of G (the radical)
- $G \rhd R_i$ for every i, and R_i/R_{i+1} is vector space
- The factor G/R can be represented effectively.

Radical factor

Since G/R has no solvable normal subgroups, it has a very special structure:

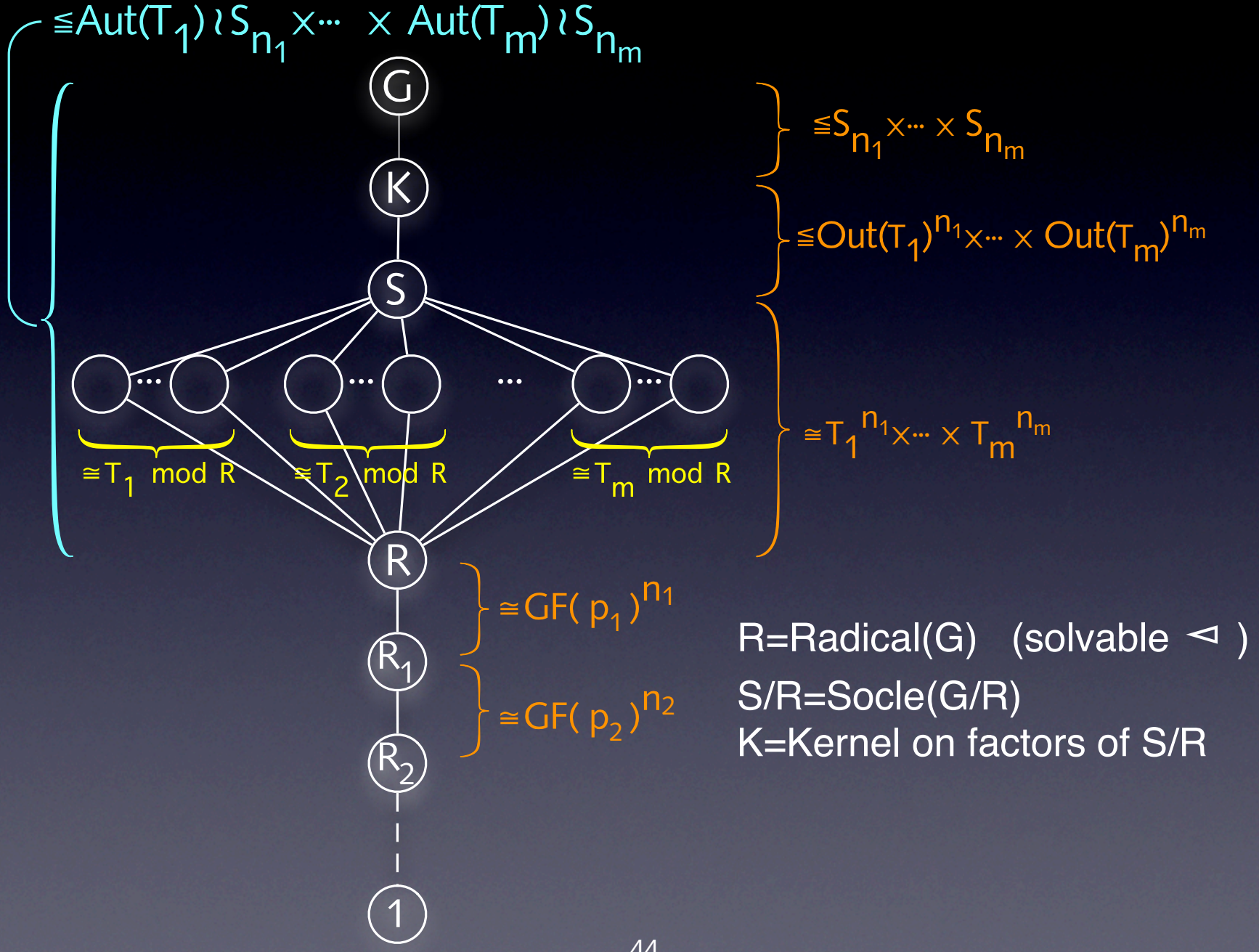
It is an iterated subdirect product (fibre product) of groups of the form

$$T^n \leq A \leq \mathrm{Aut}(T) \wr S_n$$

If we can represent $\mathrm{Aut}(T)$, we can represent A (and thus G/R) with a storage size factor n.

Normal subgroups: $\mathrm{Soc}(G/R) < K < G/R$, where K is the kernel of the S_n-projection.

Structure Summary



R=Radical(G) (solvable ◅)

S/R=Socle(G/R)

K=Kernel on factors of S/R

The radical

Since R is solvable, we typically find short orbits (on objects stabilized by R′ or R′…′) and can form an effective stabilizer chain.

An algorithm by Sims computes for such groups a stabilizer chain as well as a *polycyclic generating set (PCGS).*

Such a generating set is a concatenation of bases for the chief factors R_i/R_{i+1}. The coefficient vectors are modulo multiple primes, and arithmetic on these vectors is more complicated (collection process).

New subgroup data

While we can in principle evaluate the homomorphism $G \to G/R$ (actually we construct G/R by evaluating it for generator images) doing so in general is expensive.

Instead, we represent a subgroup U by:

- Generators of $U \cap R$, (in terms of the PCGS).
- Generators of U outside R, together with their images in G/R.
- Transition to G/R is by shadowing arithmetic with generators in U in images in the factor.

Possibilities

With such a data structure, algorithms (almost) exist to compute class representatives (elements and subgroups) and much more.

General paradigm: Calculate in G/N, then lift to G, reducing to a calculation in N. Iterating, assume N is vector space: Linear Algebra.

The limiting factors are:

- Information about simple groups.
- Memory (Can we actually store all subgroups?)
  Think about what you really want !

Application to Orbit/Stabilizer

The orbits of a normal subgroup $N \triangleleft G$ form blocks. (Converse not always true, but often.)

Thus, split an orbit/stabilizer into two parts:

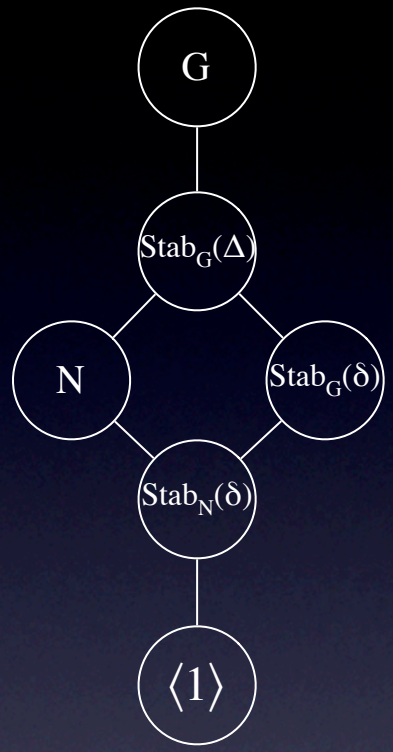
- First do Orbit/Stabilizer in the normal subgroup. Suppose Δ is the N-orbit and T the stabilizer.
- Then act with (remaining) generators for G. If the point image δ^g is not in Δ, we can add the whole image Δ^g (as it is a block).
- As the new action is on blocks, we need only one point from each image of Δ.

Generating the Stabilizer

The action on blocks determines generators for $\mathrm{Stab}_G(\Delta)$.

By mapping images back into Δ, we can correct generators by N to get $\mathrm{Stab}_G(\delta)$.

Reduces to N-orbit and G/N-orbit.



Often, instead of images, of Δ we can consider action of G/N to get $\mathrm{Stab}_G(\Delta)$.Then only correction step is needed.

Notes

We can apply this repeatedly for all normal subgroups R_i within the radical.

Ditto for socle S, permutation kernel K and derived subgroup in the radical factor.

If the group is simple, such a reduction is not available.

But (at least in principle) we know the group. A lot of information is available in theory (e.g. about elements, subgroups).

Using Elements

Given the data structure, we can compute conjugacy class representatives (and centralizers) for G.

By considering fixpoints of conjugacy class representatives, we might be able to find a cyclic subgroup U that stabilizes an object δ.

Calculate **partial** orbits ω^G and δ^G. If there is an overlap (Birthday paradox!) conjugate U so that it stabilizes ω.

Using Subgroups

If we can determine the (partial) subgroup structure of G, the following improvements are possible: (Caveat: Not all details worked out, not so much available in implementations)

- Determine stabilizer in a subgroup (e.g. maximal subgroup).
- Given a subgroup T of the stabilizer, find all minimal supergroups $U > T$ and test whether any stabilize. (Needs subgroups of simples).
- If action is primitive, stabilizer must be maximal subgroup - test whether they stabilize "similar" objects. Map.

Closing Remarks

How to use all of this?

Use of a system

Implementation of Schreier-Sims and Backtrack is feasible for an individual or a small group.

Implementation of composition series/ radical approach and its consequences probably not - Lots of group theory needed, in particular in matrix case.

Reduction to simple groups and subgroup calculations are elaborate algorithms.

Consider use of a computer algebra system.

GAP

GAP is a free and open system for computational algebra and combinatorics.

Convenient programming language.

Interpreted, but time critical routines in C.

Many group theoretic algorithms already implemented.

Package structure for user extensions.

Talk to me if you want implementation help or hints.

GAP in Teaching

I am working on a project to improve GAP for use in undergraduate algebra courses.

- Standard installers and Shell (Mac and Win)
- Education specific functionality
- A “How to do it in GAP” book, including handouts, homework. (First draft online.)
- Initially mainly algebra, but similar areas are welcome.
- I'd welcome if this became a general “education” repository.

Ceterum

Computational group theory started with **concrete problems** (e.g. from the classification of finite simple groups).

Personally, I think that algorithm development without input from applications degenerates to an exercise in logic.

I'd be most interested to hear what problems you'd like to solve.