Notes on Computational Group Theory

Alexander Hulpke

Spring 2008

Alexander Hulpke
Department of Mathematics
Colorado State University
1874 Campus Delivery
Fort Collins, CO, 80523

Chapter

I

Basics

I.1 What is Computational Group Theory

Computational Group Theory (CGT) is the study of algorithms for groups. It aims to produce algorithms to answer questions about concrete groups, given for example by generators or as symmetries of a certain algebraic or combinatorial structures.

Interest in this comes from (at least) three areas:

- Interest in developing algorithms: Can we actually calculate the objects we define theoretically in our algebra courses?
- Concrete questions about concrete groups: We are interested in a particular group and we want to find out more about it. Early examples of this happened in the classification of the finite simple groups, when group theorists predicted the existence of certain groups and then a lot of effort was needed to construct these groups and determine their properties.

  Of course users here are not restricted to group theorists. For example a chemist might want to find out some properties of the symmetry group of a differential equation, in the same way as she would use Maple to solve an integral.
- Complexity theory (which is a somewhat surprising area to come up). **The** famous problem in theoretical computer science is the question whether P=NP, i.e. whether for any problem for which we can *verify* a solution quickly (quickly here means: "polynomial runtime") we also can *find* a solution quickly. (This is one of the Millennium problems for whose solution $\$10^6$ have been on offer.) Typical cases of this are "puzzle" type problems, such as the "Traveling Salesman" problem. One particular intriguing problem of this kind is "Graph Isomorphism", i.e. the question whether two graphs, given by their adjacency matrix, are in fact isomorphic. This problem seems to lie "between P and NP" and thus might be a good bellwether for determining the relation between these problem classes.

  A graph isomorphism however is simply a permutation of the vertices, preserving the edge incidences. Thus there has been the hope that permutation group methods can help in studying this problem.

  Indeed in 1982, G. Luks[Luk82] solved the problem in polynomial time for a particular class of graphs. His solution uses substantial (computational) group theory. Since then there has been much interest in CGT from theoretical computer science.

This course is intended as an introduction to computational group theory which will lead you to a level where you could starting to read the research literature. The textbook by Holt [HEO05] is a good starting point, covering the material of these notes and much more.

I.2 A short introduction to GAP

Some computational group theory methods are implemented in many computer algebra systems, but there are two systems which specialize on it: GAP and Magma. We will use GAP.

See the first homework sheet.

I.3 Memory

As often in computation we could buy runtime at the expense of memory. In fact for many larger calculations memory use is more of an obstacle than run time. (You can wait a day longer but this still won't increase your systems memory.)

To understand some of the choices or trade-offs we will be making, it is useful to understand a bit about memory use for different objects. The numbers given are for GAP on a 32-bit system; other implementations wil face essentially similar issues.

Numbers: A computer stores numbers in base 2, so we need $2 \cdot \log_2(n)$ bits to represent a signed number of magnitude n. (In fact we typically allocate memory in chunks of 4 bytes on a 32 bit system.)

Small Numbers: All processors have built in arithmetic for small numbers (up to 32 bits). We will use this arithmetic for such small numbers. (In fact for technical reasons the limit in GAP is $\pm 2^{28}$. There is a notable slowdown if numbers get above 2^{28}.)

Finite field elements Unless the field is very large, we can easily squeeze them in 4 bytes per number.

Permutations A permutation on n points is simply stored as a list of images for each point. If $n \leq 2^{16}$ we can do with 2 bytes per point (and thus $2n$ bytes storage), in general we use 4 bytes per point and thus require $4n$ bytes of memory. (To simplify arithmetic we usually do not throw away trailing fix points. I.e. the identity element of S_{10} is stored internally as images of 10 points. Internal magic makes this invisible to the user.)

Matrices are simply lists of vectors, every vector again being a list. (GAP also uses compact types for matrices over finite fields.)

To put these numbers into context, suppose we have a permutation group acting on 1000 points. Then each permutation takes 2kB of memory. 500 permutations take 1MB. On a modern machine (2GB) we could thus store about 1 million permutations if we used up all memory. On the other hand if we have degree 100000, we could only store 500 permutations.

As we want to be able to work with such groups we clearly are only able to store a small proportion of group elements.

I.4 Orbits and Stabilizers

Group Actions

One of the most prominent uses of groups is to describe symmetries of objects. Thus it should not surprise that some fundamental algorithms deal with group actions. (Indeed, the algorithms in this section are the only basic algorithms which specifically use that one is working with groups.)

A group G acts[1] on a set Ω if

- $\omega^1 = \omega$ for all $\omega \in \Omega$
- $(\omega^g)^h = \omega^{gh}$ for all $\omega \in \Omega$, $g, h \in G$.

In this case we define for $\omega \in \Omega$ the *Orbit* $\omega^G = \{\omega^g \mid g \in G\} \subset \Omega$ and the *Stabilizer* $\mathrm{Stab}_G(\omega) = \{g \in G \mid \omega^g = \omega\} \leq G$.

Lemma I.1: There is a bijection between ω^G and the set $\mathrm{Stab}_G(\omega)\backslash G$ (i.e. right cosets of $\mathrm{Stab}_G(\omega)$ in G), given by

$$\omega^g \leftrightarrow \mathrm{Stab}_G(\omega) \cdot g$$

In particular $\left|\omega^G\right| = \left[G : \mathrm{Stab}_G(\omega)\right]$.

If G acts on Ω, we get an homomorphism $\varphi : G \to S_{|\Omega|}$, we call this the *action homomorphism*.

By the properties of group actions we have that $\delta^g \in \omega^G$ for every $\delta \in \omega^G$ and every $g \in G$.

[1] In these notes we will always have groups acting on the right, and consider right cosets and row vectors. We will also write homomorphisms like exponentiation on the right, alluding to the fact that the action of an automorphism group is a group action.

Computing an orbit

In general we only have generators of G, not all group elements. To calculate all images of a point $\omega \in \Omega$, we use the fact that every element of G can be expressed as product of generators and their inverses.

Note I.2: If G is finite, we can express for each $g \in G$ its inverse g^{-1} as positive exponent power of g. We therefore make the following assumption:

> If G is not know to be finite, we assume that the generating set of G contains for each generator also its inverse.

With this convention we can assume that every element of G is the product of generators of G.

The following lemma then gives the basic idea behind the orbit algorithm

Lemma I.3: Let $G = \left\langle \underline{g} \right\rangle$ with $\underline{g} = \{g_1, \ldots, g_m\}$ and let $\omega \in \Omega$ and $\Delta \subset \Omega$ such that

a) $\omega \in \Delta$

b) For all $\delta \in \Delta$ and every generator g_i we have that $\delta^{g_i} \in \Delta$

c) For every $\delta \in \Delta$ there exists a sequence of indices $i_1, \ldots, i_k$ such that $\delta = (\cdots(\omega^{g_{i_1}})^{g_{i_2}}\cdots))^{g_{i_k}}$

Then $\omega^G = \Delta$

Proof: By property a) and c) we have that every $\delta \in \Delta$ is in ω^G. On the other hand property b) shows that Δ must be a union of orbits. □

This gives the following algorithm:

Algorithm I.4: The "plain vanilla" orbit algorithm.

Input: A group G, given by a generating set $\underline{g} = \{g_1, \ldots, g_m\}$, acting on a domain Ω. Also a point $\omega \in \Omega$.

Output: return the orbit ω^G.

```
begin
 1: Δ := [ω];
 2: for δ ∈ Δ do
 3:    for i ∈ {1,...,m} do
 4:       γ := δ^{g_i};
 5:       if γ ∉ Δ then
 6:          Append γ to Δ;
 7:       fi;
 8:    od;
 9: od;
10: return Δ;
```

end

Note that the **for**-loop in line 2 runs also through elements added to *orb* in the course of the algorithm.

Note I.5: Instead of starting with ω we could start with multiple points and then calculate the union of orbits containing these points.

Note I.6: In the algorithm, we compute the image of every orbit element under every group generator. If we do not only test whether $\gamma \in \Delta$, but identify the position of $\gamma \in \delta$ we obtain the permutation image of G. In the same way we can evaluate action homomorphisms.

Performance I.7: If we have m generators and an orbit of length n there will be mn images to compute. The cost of each image will depend on the actual action, but is proportional to the number of images.

On the other hand the test $\gamma \in \Delta$ in line 6 is essentially a search problem. As soon as the time for such a test is not constant (even a binary search in a sorted list of length n is $\mathcal{O}(\log(n))$) this test will eventually dominate the run time of the algorithm. It therefore is worth devoting extra data structures (e.g. a sorted list of the elements, or a hash table for a suitably defined hash key) towards reducing the cost of this test.

An easy consequence of the orbit algorithm is that we can obtain all elements of a group G by computing the orbit of 1 under the action of G by right multiplication. In particular, we could test in an extremely crude way whether an element is in a group. (In general we want to do **much** better.)

Representatives

In many applications we do not only want to find the orbit of ω but also find for $\delta \in \omega^G$ an element $g \in G$ such that $\omega^g = \delta$.

We do this by calculating such elements for every orbit element. Such a list of representatives is called a *transversal*. By lemma I.1 it simultaneously is a set of representatives for the cosets of $\mathrm{Stab}_G(\omega)$.

At this point it makes sense to consider ω^G as a list (with fixed ordering) to maintain a correspondence between orbit points and corresponding transversal elements.

To simplify notation, we will simply index a transversal with orbit elements: $T[\delta]$. By this we mean $T[i]$ where $\Delta[i] = \delta$. (Again, as in performance remark I.7 this lookup might come at a nontrivial cost and merits special consideration in an implementation.)

Note I.8: What about mapping δ to γ for arbitrary $\delta, \gamma \in \omega^G$? We simply find g, h such that $\omega^g = \delta$ and $\omega^h = \gamma$, then $\delta^{g^{-1}h} = \gamma$

For building the list of representatives we now just observe that if x is a representative for δ, then xg is a representative for δ^g. This gives the following modification of the orbit algorithm:

Algorithm I.9: Orbit algorithm with transversal computation

Input: A group G, given by a generating set $\underline{g} = \{g_1, \ldots, g_m\}$, acting on a domain Ω. Also a point $\omega \in \Omega$.

Output: return the orbit ω^G and a transversal T.

begin

```
 1: Δ := [ω];
 2: T := [1];
 3: for δ ∈ Δ do
 4:    for i ∈ {1,...,n} do
 5:       γ := δ^{g_i};
 6:       if γ ∉ Δ then
 7:          Append γ to Δ;
 8:          Append T[δ]·g_i to T;
 9:       fi;
10:    od;
11: od;
12: return  Δ, T;
```

end

Note I.10: It is worth observing that the representative $T[\delta]$ obtained in this algorithm is a *shortest* product of group generators that has the desired mapping. If we use the orbit algorithm to obtain all group elements, we can therefore obtain a *minimal* factorization for all group elements, however at high memory cost.

In fact any known algorithm that guarantees a minimal factorization eventually reduces to a brute-force enumeration similar to this algorithm. Improvements are possible towards reducing the storage required for each group element, but this only gains a constant factor. Heroic parallelizations have been used for example to bound the maximum number of moves for Rubik's cube [KC07, Rok08].

Schreier Vectors

If you think a bit about the previous algorithm, you will notice a big problem: We store one group element for every element in the orbit. In general group elements take much more storage than orbit elements, so memory requirements quickly get problematic for longer orbits.

To avoid memory overflow, we will be using the following idea:

Definition I.11: Let $\Delta = \omega^G$ (again considered as a list). A *Schreier vector* (or *factored transversal*) is a list S of length $|\Delta|$ with the following properties:

- The entries of S are generators of G (or the identity element). (In fact the entries are *pointers* to generators, thus requiring only one pointer per entry instead of one group element.)

- $S[\omega] = 1$
- If $S[\delta] = g$ and $\delta^{g^{-1}} = \gamma$ then γ precedes δ in the orbit.

We can compute a Schreier vector easily by initializing $S[\omega] = 1$. In the orbit algorithm, we then set $S[\delta] := g$ whenever a new point δ is obtained as image $\delta = \gamma^g$ of a known point γ.

Schreier vectors can take the place of a transversal:

ALGORITHM I.12: If S is a Schreier vector for a point $\omega \in \Omega$, the following algorithm computes for $\delta \in \omega^G$ a representative r such that $\omega^r = \delta$.

begin
1: $\gamma := \delta$;
2: $r := 1$;
3: **while** $\gamma \neq \omega$ **do**
4: $g := S[\gamma]$;
5: $r := g \cdot r$;
6: $\gamma = \gamma^{g^{-1}}$;
7: **od**;
8: **return** r;
end

Proof: The algorithm terminates by condition 3 for a Schreier vector. Also notice that we always have that $\gamma^r = \delta$. Thus when the algorithm terminates (which is for $\gamma = \omega$) the result r has the desired property. □

NOTE I.13: In practice it makes sense to store not generators, but their inverses in the Schreier vector. This way we do not need to repeatedly invert elements in step 6. Then r is computed by forming the product of these inverse generators *in reverse order* (i.e. in step 5 forming the product $r \cdot (g^{-1})$) and inverting the final result: If $r = fgh$ then $r = (h^{-1}g^{-1}f^{-1})^{-1}$.

PERFORMANCE I.14: To keep runtime short it is desirable that the number of products needed for each representative r is small. (This is called a *shallow Schreier tree*.) An example of a bad case is the group generated by the n-cycle $(1, 2, \ldots, n)$. Here $n-1$ multiplications are needed to obtain the representative for n in the orbit of 1.

To avoid such bad situations, one can modify the definition order of new points in the orbit algorithm. It also helps to adjoin extra (random) generators. More details can be found in [Ser03, Sec.4.4].

NOTE I.15: Unless $|\omega^G|$ is very small, we will use Schreier vectors instead of a transversal and will use algorithm I.12 to obtain (deterministic!) corresponding representatives. To simplify algorithm descriptions, however we will just talk about transversal elements with the understanding that a transversal element $T[\delta]$ is actually obtained by algorithm I.12.

Stabilizer

The second modification to the orbit algorithm will let us determine a generating set for the stabilizer $\mathrm{Stab}_G(\omega)$. The basis for this is the following lemma that relates group generators and a set of fixed coset representatives to subgroup generators.

LEMMA I.16: (SCHREIER) Let $G = \langle \underline{g} \rangle$ a finitely generated group and $S \leq G$ with $[G:S] < \infty$. Suppose that $\underline{r} = \{r_1, \ldots, r_n\}$ is a set of representatives for the cosets of S in G, such that $r_1 = 1$. For $h \in G$ we write $\bar{h}$ to denote the representative $\bar{h} := r_i$ with $Sr_i = Sh$. Let

$$U := \{r_i g_j (\overline{r_i g_j})^{-1} \mid r_i \in \underline{r}, g_j \in \underline{g}\}$$

Then $S = \langle U \rangle$. The set U is called a set of *Schreier generators* for S.

Proof: As $S \cdot (r_i g_j) = S\overline{r_i g_j}$ by definition of $\bar{\cdot}$, we have that $U \subset S$.

We thus only need to show that every $x \in S$ can be written as a product of elements in U. As $x \in G = \langle \underline{g} \rangle$ we can write $x = g_{i_1} g_{i_2} \cdots g_{i_m}$ with $g_{i_j} \in \underline{g}$. (Again, for simplicity we assume that every element is a product of generators with no need for inverses.)

We now *rewrite* x iteratively. In this process we will define a set of elements $t_i \in \underline{r}$ which are chosen from the fixed coset representatives:

$$\begin{aligned}
x &= g_{i_1} g_{i_2} \cdots g_{i_m} \\
&= t_1 g_{i_1} g_{i_2} \cdots g_{i_m} \quad [\text{setting } t_1 := r_1 = 1] \\
&= t_1 g_{i_1} ((\overline{t_1 g_{i_1}})^{-1} \cdot \overline{t_1 g_{i_1}}) g_{i_2} \cdots g_{i_m} \quad [\text{insert } 1] \\
&= (t_1 g_{i_1} (\overline{t_1 g_{i_1}})^{-1}) t_2 g_{i_2} \cdots g_{i_m} \quad [\text{set } t_2 := \overline{t_1 g_{i_1}}] \\
&= \underbrace{t_1 g_{i_1} (\overline{t_1 g_{i_1}})^{-1}}_{=:u_1 \in U} t_2 g_{i_2} \left((\overline{t_2 g_{i_2}})^{-1} \cdot \overline{t_2 g_{i_2}} \right) \cdots g_{i_m} \\
&= u_1 \cdot \underbrace{t_2 g_{i_2} \overline{t_2 g_{i_2}}^{-1}}_{=:u_2 \in U} \cdot t_3 g_{i_3} \cdots g_{i_m} \quad [\text{set } t_3 = \overline{t_2 g_{i_2}}] \\
&\vdots \\
&= u_1 u_2 \cdots \cdot u_{m-1} \cdot t_m g_{i_m}
\end{aligned}$$

In this process t_j is the coset representative for $g_{i_1} \cdots g_{i_{j-1}}$ (easy induction proof). Thus $\overline{t_m g_{i_m}} = 1$, as $x \in S$. Thus $t_m g_{i_m} = t_m g_m (\overline{t_m g_{i_m}})^{-1} \in U$ which gives an expression of x as product of elements in U. □

In our application we have $S = \mathrm{Stab}_G(\omega)$ and we can use the elements of a transversal for ω as coset representatives. (The representative for the coset Sg is $T[\omega^g]$.)

We thus get the following modification to the orbit algorithm:

ALGORITHM I.17: Orbit/Stabilizer algorithm

Input: A group G, given by a generating set $\underline{\boldsymbol{g}} = \{g_1, \ldots, g_m\}$, acting on a domain Ω. Also a point $\omega \in \Omega$.

Output: return the orbit ω^G, a transversal T, and the stabilizer $S = \mathrm{Stab}_G(\omega)$.

begin

1: $\Delta := [\omega]$;
2: $T := [1]$;
3: $S := \langle 1 \rangle$;
4: **for** $\delta \in \Delta$ **do**
5: **for** $i \in \{1, \ldots, n\}$ **do**
6: $\gamma := \delta^{g_i}$;
7: **if** $\gamma \notin \Delta$ **then**
8: Append γ to Δ;
9: Append $T[\delta] \cdot g_i$ to T;
10: **else**
11: $S := \langle S, T[\delta] \cdot g_i \cdot T[\gamma]^{-1} \rangle$;
12: **fi**;
13: **od**;
14: **od**;
15: **return** Δ, T, S;

end

NOTE I.18: We have not described how to compute the closure in step 11. The most naive version would be to simply accumulate generators, typically redundant generators (i.e. elements already in the span of the previous elements) are discarded if an efficient element test for subgroups exists (e.g. section II.1 in chapter II).

PERFORMANCE I.19: We will see later (note III.33 in chapter III) that the rather large number of Schreier generators $\left|\underline{\boldsymbol{g}}\right| \cdot [G{:}S]$ in general is the best possible for a subgroup generating set.

However in practice this set of generators is typically highly redundant. We can remove obvious redundancies (duplicates, identity), but even then much redundancy remains.

There are essentially three ways to deal with this:

- For every arising Schreier generator, we test in step 11 whether it is already in the subgroup generated by the previous Schreier generators and discard redundant generators. Doing so requires many element tests.
- We pick a small (random) subset of the Schreier generators and hope[2] that these elements generate the stabilizer. To make this deterministic (i.e. repeat if it fails) one needs a means of verification that everything went well.

[2] The probability that a small random subset generates a finite group is often very high. Proofs exist for example for random subsets of two elements in the case of symmetric groups [Dix69] or simple groups [LS95].

A more concrete analysis of generation probability (which makes it possible to make the probability of an error arbitrary small) is possible if one chooses *random subproducts* of the Schreier generators (essentially products of the form $s_1^{\varepsilon_1} s_2^{\varepsilon_2} \cdots s_k^{\varepsilon_k}$ with $\varepsilon_i \in \{0, \pm 1\}$) [BLS97]. Still, the verification problem remains.

Application: Normal Closure

Let $U \leq G$. The *normal closure* of U in G is

$$\langle U \rangle_G = \bigcap \{N \mid U \leq N \lhd G\}$$

the smallest normal subgroup of G containing U.

One of its uses is in the computation of commutator subgroups, for example if $G = \left\langle \underline{\boldsymbol{g}} \right\rangle$, then $G' = \left\langle a^{-1}b^{-1}ab \mid a, b \in \underline{\boldsymbol{g}} \right\rangle_G$.

If we start with generators of U and G, we can compute this closure in a variant of the orbit algorithm:

ALGORITHM I.20: NormalClosure of a subgroup.

Input: Two generating systems $\underline{\boldsymbol{g}}$ and $\underline{\boldsymbol{u}}$ for subgroups $G = \left\langle \underline{\boldsymbol{g}} \right\rangle$ and $U = \langle \underline{\boldsymbol{u}} \rangle$.

Output: A generating system for the normal closure $\langle U \rangle_G$.

begin

1: $\underline{\boldsymbol{n}} := [\,]$;
2: **for** $x \in \underline{\boldsymbol{u}}$ **do** {start with $\underline{\boldsymbol{u}}$}
3: Add x to $\underline{\boldsymbol{n}}$;
4: **od**;
5: **for** $d \in \underline{\boldsymbol{n}}$ **do** {orbit algorithm starting with $\underline{\boldsymbol{n}}$}
6: **for** $g \in \underline{\boldsymbol{g}}$ **do**
7: $c := d^{\bar{g}}$;
8: **if** $c \notin \langle \underline{\boldsymbol{n}} \rangle$ **then** {inclusion in group closure}
9: Add c to $\underline{\boldsymbol{n}}$;
10: **fi**;
11: **od**;
12: **od**;
13: **return** $\underline{\boldsymbol{n}}$;

end

Proof: The algorithm clearly terminates, if G is finite, as only finitely many elements may be added to $\underline{\boldsymbol{n}}$ in step 8.

As $\underline{\boldsymbol{n}}$ is initialized by $\underline{\boldsymbol{u}}$, we have that $U \leq \langle \underline{\boldsymbol{n}} \rangle$. Furthermore, as we only add conjugates of the elements in $\underline{\boldsymbol{n}}$, we have that $\langle \underline{\boldsymbol{n}} \rangle \leq \langle U \rangle_G$.

We now claim that for every $x \in \langle \underline{\boldsymbol{n}} \rangle$ and every $g \in G$ we have that $x^g \in \langle \underline{\boldsymbol{n}} \rangle$. As $(xy)^g = x^g y^g$ it is sufficient to consider $x \in \underline{\boldsymbol{n}}$. Because we can express g as a word in $\underline{\boldsymbol{g}}$ this statement holds by the same argument as in the orbit algorithm. This proves that $\langle \underline{\boldsymbol{n}} \rangle \lhd G$. But $\langle U \rangle_G$ is the smallest normal subgroup of G containing U,

which proves that $\langle \underline{\boldsymbol{n}} \rangle = \langle U \rangle_G$. □

I.5 Random Elements

We have already talked (and will talk again) about using random elements. In this section we want to describe a general algorithm to form (pseudo)-random elements of a group $G = \left\langle \underline{\boldsymbol{g}} \right\rangle$ if only the generating set $\underline{\boldsymbol{g}}$ is known.

Our first assumption is that we have a (perfect) random number generator. Using this, one can try to multiply generators together randomly. The problem is that if we only multiply with generators, the word length grows very slowly, making it difficult to obtain any kind of equal distribution in a short time.

This is resolved by multiplying products of elements together iteratively. The following algorithm is a modification of [CLGM+95] due to Charles Leedham-Green. It looks deceptively simple, but performs in practice rather well and its behaviour has been studied extensively [GP06]. Unfortunately there are cases when its result will not approximate a uniform distribution [BP04].

ALGORITHM I.21: (Pseudo)Random, "Product Replacement"

Let $\underline{\boldsymbol{g}}$ a set of group elements. This algorithm returns pseudo-random elements of $\left\langle \underline{\boldsymbol{g}} \right\rangle$.

The algorithm consists of an initialization step and a routine that then will return one pseudo-random group element in every iteration.

The routine keeps a (global) list X of $r = \max(11, \left|\underline{\boldsymbol{g}}\right|)$ group elements and one extra group element a. (Experiments show that one needs $r \geq 10$.)

PSEUDORANDOM()

begin

1: s := RANDOM($[1..r]$);{pick two random list elements}
2: t := RANDOM($[1..r] \setminus [s]$);
3: e := RANDOM($[-1, 1]$);{random choice of product/quotient}
4: **if** RANDOM($[1, 2]$) = 1 **then** {random product order}
5: $X[s] := X[s]X[t]^e$;{replace one list entry by product}
6: $a := aX[s]$;{accumulate product}
7: **else**
8: $X[s] := X[t]^e X[s]$;{replace one list entry by product}
9: $a := X[s]a$;{accumulate product}
10: **fi**;
11: **return** a;

end

The list X is initialized by the following routine:

begin

$X = []$;{initialize with repetitions of the generator set}
$k := \left|\underline{\boldsymbol{g}}\right|$;
for $i \in [1..k]$ **do**
$X[i] := g_i$;
od;
for $i \in [k+1..r]$ **do**
$X[i] := X[i-k]$;
od;
$a := 1$;
for $i \in [1..50]$ **do** {50 is heuristic}
PSEUDORANDOM(); {Initial randomization}
od;

end

The iterated multiplication in steps 5/6 and 8/89 of the PSEUDORANDOM routine ensures a quick growth of word lengths.

I.6 How to do it in GAP

Group Actions

Group actions being a fundamental functionality, GAP has a rather elaborate setup for group actions. The heart of it is to specify the actual action by a function: `actfun(ω,g)`, which will return the image ω^g for the particular definition of the action. No[3] test is performed that the function actually implements a proper group action from the right. GAP comes with a couple of predefined actions:

`OnPoints` Calculates the image as calculated by the caret operator `^`. For example permutations on points, or conjugacy in a group. If no action function is given, the system defaults to `OnPoints`.

`OnTuples` Acts on lists of points, acting with the same element on each entry separately via `OnPoints` (i.e. the induced action on tuples).

`OnSets` Works like `OnTuples` but the resulting lists of images is sorted, considering [B,A] equal to [A,B] (i.e. the induced action on sets of points).

`OnRight` The image is the image under right multiplication by the group element. For example matrices on row vectors or group elements on cosets. The action group on the cosets of a subgroup by right multiplication is so important, that GAP provides special syntax to do this efficiently (i.e. without need to store cosets as special objects, in many cases even without the need to store an explicit list of coset representatives). In a slight abuse of notation this is achieved by the command

[3] well, almost no

`ActionHomomorphism(G,RightTransversal(G,S),OnRight);`

`OnLines` is used to implement the projective action of a matrix group on a vector space: Each 1-dimensional subspace $\langle v\rangle$ of the row space is represented by a vector $w = c \cdot v$ scaled such that the first nonzero entry of w is one.

Using actions specified this way, one can now calculate

`Orbit(`G`,`ω`,actfun);`

`RepresentativeAction(`G`,`ω`,`δ`,actfun);`,

`Stabilizer(`G`,`ω`,actfun);`,

`ActionHomomorphism(`G`,`Ω`,actfun,"surjective");` returns a homomorphism from G to the permutation action on Ω (a list of points whose arrangement is used to write down permutations). The extra argument `"surjective"` ensures that the range is set equal to the image (otherwise the range is $S_{|\Omega|}$). If only the image of this homomorphism is desired, one can use the function `Action` instead.

It should be stressed that with few exceptions (Permutation groups on points, sets or tuples, groups on their elements by conjugation) these functions default to the fundamental algorithms described in this chapter. In particular their run time and memory use is proportional to the length of the orbit. Action homomorphisms use permutation group machinery to compute preimages.

Variations

As described in note I.7 the bottleneck of all these algorithms is finding points in the partial orbit, both to check whether they are new, and to identify corresponding transversal elements. To do so efficiently, it is useful to know the domain Ω in which the orbit lies: Ω might be small and afford a cheap indexing function – in this case the position in Ω can be used for lookup. Alternatively, Ω can give information about what kind of hash function to use. For example, when acting on vectors in characteristic 2, calculating the orbit of $[\bar{1}, \bar{0}, \ldots, \bar{0}]$ does not specify, whether all other vectors in the orbit actually are defined over GF(2) or if they only are defined over extension fields[4].

All action functions therefore take (a superset of) the domain as an optional second argument, e.g. `Orbit(`G`,`Ω`,`ω`,actfun);`. Doing so can speed up the calculation.

A second variant (relevant for finding representatives or calculating stabilizers) is the situation that G acts via a homomorphism, for example if a permutation

[4] As this might actually depend on the user-supplied function `actfun` the system **cannot** do this in general!

group acts on a module via matrices. In such a situation we do not want to actually evaluate the homomorphism at each step. On the other hand the only group elements ever acting are the generators. It therefore is possible to specify two lists, generators $\underline{\boldsymbol{g}}$ and their acting homomorphic images $\underline{\boldsymbol{h}}$ as optional arguments. For example in the function call `Stabilizer(`G`,`ω`,`$\underline{\boldsymbol{g}}$`,`$\underline{\boldsymbol{h}}$`,actfun);`

Then images are calculated using $\underline{\boldsymbol{h}}$, but transversal elements and stabilizer generators calculated using $\underline{\boldsymbol{g}}$, i.e. as elements of G.

Random elements

The product replacement algorithm, as described in this chapter, is implemented why the function `PseudoRandom`. There also is a function `Random`, which guarantees[5] a random distribution of group elements. This function essentially uses methods to enumerate all group elements, and simply returns a group element for a random index number.

[5] assuming – which is not true – that the underlying random number generator creates a true random distribution

Chapter

II

Permutation Groups

Sediento de saber lo que Dios sabe,
Judá León se dio a permutaciones
de letras y a complejas variaciones
Y al fin pronunció el Nombre que es la Clave,

La Puerta, el Eco, el Huésped y el Palacio.

El Golem
Jorge Luis Borges

Probably the most important class of groups are permutation groups, not least because every finite group can be represented this way. If you are interested in details, there is a monograph [Ser03] dedicated to algorithms for such groups which goes in much more detail.

II.1 Stabilizer Chains and their Computation

We now assume that G is a (potentially large) permutation group, given by a set of permutation generators. We want to compute with this group (for example: find its order, and to have an element test), without having to enumerate (and store!) all its elements. Obviously we have to store the generators, we also are willing to store some further group elements, but in total we want to store just a few hundred elements, even if the group has size several fantastillions.

Stabilizer Chains

The algorithm we want to develop is due to Charles Sims [Sim70]. As it uses Schreier's lemma I.16 this algorithm has been known commonly as the "Schreier-Sims" algorithm.

Its basic idea is the following: We consider a list of points $B = (\beta_1, \ldots, \beta_m)$, such that the identity is the only element $g \in G$ with the property that $\beta_i^g = \beta_i$ for all i. We call such a list B a *base* for G. (It clearly is not unique.) Corresponding to the base we get a *Stabilizer chain*: This is a sequence of subgroups of G, defined by $G^{(0)} := G$, $G^{(i)} := \mathrm{Stab}_{G^{(i-1)}}(\beta_i)$. (By the definition of a base, we have that $G^{(m)} = \langle 1 \rangle$.)

One interesting property of a base is that every permutation $g \in G$ is determined uniquely by the images of a base $\beta_1^g, \ldots, \beta_m^g$ it produces. (If h produces the same images, g/h fixes all base points.)

Note II.1: In general a base is rather short (often length < 10 even for large groups) but there are obvious cases (e.g. symmetric and alternating groups) where the base is longer. Still, as every stabilizer index must be at least 2, the length of a base must be bounded by $\log_2 |G|$.

Sims' idea now is that we can describe G in terms of the cosets for steps in this chain: An element $g \in G^{(i-1)}$ will be in a coset of $G^{(i)}$. Thus we have that $g = a \cdot r$ with $a \in G^{(i)}$ and b a coset representative for $G^{(i)}$ in $G^{(i-1)}$. As $G^{(i)} = \mathrm{Stab}_{G^{(i-1)}}(\beta_i)$ these coset representatives correspond to the orbit of β_i under $G^{(i-1)}$.

By using this kind of decomposition inductively, we can write any $g \in G$ in the form $g = b_m b_{m-1} \cdots b_1$ with b_i a coset representative for $G^{(i)}$ in $G^{(i-1)}$ and thus corresponding to a point in the orbit $\beta_i^{G^{(i-1)}}$.

We can describe these orbits and sets of representatives using the orbit algorithm we studies in the last chapter.

On the computer we thus store a stabilizer chain in the following way:

Each subgroup $G^{(i)}$ in the stabilizer chain is represented by a record with entries giving

- the `generators` of $G^{(i)}$,
- the `orbit` of β_{i+1} under $G^{(i)}$ (we shall use the convention that β_{i+1} = `orbit`[1]),
- a corresponding `transversal` (which in fact will be implemented using a Schreier vector) and
- a pointer to the `stabilizer` which is the record for $\mathrm{Stab}_{G^{(i)}}(\beta_{i+1})$.

Example II.2: Let $G = A_4$ with base $[1, 2]$. Then $G = G^{(0)} = \langle (1,2,3), (2,3,4) \rangle$, $G^{(1)} = \mathrm{Stab}_G(1) = \langle (2,3,4) \rangle$ and $G^{(2)} = \mathrm{Stab}_G(1,2) = \langle \rangle$.

We thus get (for example) the following data structure:

```
rec(generators:=[(1,2,3),(2,3,4)],
    orbit:=[1,2,3,4],
    transversal:=[(),(1,2,3),(1,3,2),(1,4,2)],
    stabilizer := rec(
       generators:=[(2,3,4)],
```

```
orbit:=[2,3,4],
transversal:=[(),(2,3,4),(2,4,3)],
stabilizer:= rec(
  generators:=[])  )  )
```

NOTE II.3: How do we actually determine a base? We determine the next base point when we need it: β_i is simply chosen to be a point moved (so we have a proper orbit) by some generator of $G^{(i-1)}$.

In some applications, one also might want a base containing particular points, which we would chose first.

For such a data structure, the following algorithm produces the factorization of a group elelemnt as product of coset representatives:

ALGORITHM II.4: Let $g \in G^{(0)}$. We want to find the expression $g = b_m b_{m-1} \cdots b_1$ with b_i a coset representative for $G^{(i)}$ in $G^{(i-1)}$.

Input: A stabilizer chain C for a group G and an element $g \in G$

Output: A list $L = [b_1, b_2, \ldots, b_m]$ of coset representatives, such that $g = b_m b_{m-1} \cdots b_1$.

begin

1: $L := [\,]$;
2: **while** C.generators <> $[\,]$ **do**
3: $\beta := C$.orbit$[1]$;
4: $\delta = \beta^g$;
5: $r := C$.transversal$[\delta]$;
6: $g := g/r$;
7: Add r to L;
8: $C := C$.stabilizer;
9: **od**;
10: **return** L

end

Proof: Observe that $\beta^r = \beta^g$, thus the new g in line 6 is in the stabilizer of β. Thus at the end of the algorithm, after dividing off representatives, we must have $g = 1$. □

A naive way to calculate a stabilizer chain would be to simply compute the orbit of β_1 under $G = G^{(0)}$ and generators for $G^{(1)} = \mathrm{Stab}_{G^{(0)}}(\beta_1)$ using the Orbit/Stabilizer algorithm. We then iterate for $G^{(1)}$ until we end up with a trivial stabilizer.

The only problem with this approach is the large number of Schreier generators: In each layer the number of generators will increase by the index, leaving us about $|G|$ generators in the last step. Overal this would result in a run time that is exponential in the number of points.

Element Test

The remedy for this problem lies in the fact that the a stabilizer chain lets us also do an element test for the group represented by the chain. This is done by a small modification of the algorithm used to factor into coset representatives.

Consider what happens in algorithm II.4 if $g \notin G$. Then obviously the algorithm cannot terminate with $g = 1$. Instead what will happen is that at some iteration the image δ may not be in the orbit of β. (This might be at the very end of the algorithm where .generators and .transversal are empty.

If we check for this situation, we get a test for whether an element is in a permutation group described by a stabilizer chain. We call this resulting procedure "ElementTest(C, a)". This process also is sometimes called "*sifting*".

ALGORITHM II.5: Same setup as algorithm II.4, but if the element is not is the group, an error is returned.

begin

1: $L := [\,]$;
2: **while** C.generators <> $[\,]$ **do**
3: $\beta := C$.orbit$[1]$;
4: $\delta = \beta^g$;
5: **if** C.transversal$[\delta]$ does not exist **then**
6: **return** not contained;
7: **fi**;
8: $r := C$.transversal$[\delta]$;
9: $g := g/r$;
10: Add r to L;
11: $C := C$.stabilizer;
12: **od**;
13: **if** $g \neq ()$ **then**
14: **return** not contained;
15: **else**
16: **return** L
17: **fi**;

end

To prove correctness of a stabilizer chain computation the following observations will be useful:

DEFINITION II.6: A *partial stabilizer chain* is a data structure as described for a stabilizer chain such that on each layer C we have that for the base point $\beta = C$.orbit$[1]$ the orbit of β under $\langle C$.generators$\rangle$ is C.orbit and that

$$\mathrm{Stab}_{\langle C.\mathtt{generators}\rangle}(\beta) \geq \langle C.\mathtt{stabilizer.generators}\rangle$$

If equality holds on every layer, the partial stabilizer chain is called *proper*.

LEMMA II.7: Let C be a layer of a partial stabilizer chain with orbit starting at $\beta =$

C.orbit[1] and $G = \langle C.\texttt{generators}\rangle$. Then C is a (proper) stabilizer chain for G if any of the following conditions hold.[1]

1. C.stabilizer is a proper stabilizer chain for $\mathrm{Stab}_G(\beta)$.
2. $|G| = |C.\texttt{orbit}| \cdot |\langle C.\texttt{stabilizer}\rangle|$

The Schreier-Sims algorithm

The element test gives us the chance to remove redundant Schreier generators: We will build the stabilizer chain not layer by layer, accumulating a large number of Schreier generators, but instead after obtaining one Schreier generator first test whether it is redundant by checking whether it is contained in the span of the span of the Schreier generators found so far. The whole stabilizer chain is computed by starting with the chain for a trivial group, and adding the groups generators, one by one, as if they were Schreier generators from a higher level.

To do an element test with the existing partial data structure, we assume that the layer below the one in which we are calculating orbits (i.e. the C.stabilizer layer) is a proper stabilizer chain. Whenever we modify a layer, we will have to ensure that it is a a proper chain, before returning back.

Once we enforce this condition, lemma II.7 ensures that at the end of the calculation the whole stabilizer chain is correct.

We now describe the processing of a new (Schreier) generator a which is given to a layer C in the chain. We first use the element test from algorithm II.5 to check whether a is contained in the group described by C. (Remember, that we assume that C is a proper chain, if we pass generators to it.) If a is contained, it is redundant, and we ignore it.

Otherwise we know that C does not describe the correct stabilizer in the group, but only a subgroup. We therefore need to add a to the generators of C and expand the orbit accordingly (i.e. calculate images of all orbit elements under a and – if any new orbit elements arose – calculate images for these under all the generators) to ensure that C is a partial stabilizer chain. Newly arising Schreier generators are fed (in a recursive call) to the next layer C.stabilizer.

(If the element test for a did fail not on layer C, but on a lower layer D, this process immediately creates Schreier generators.)

Once this orbit extension (and processing of Schreier generators) is complete we know that C.stabilizer is the proper stabilizer for layer C. By lemma II.7, this means that C is a proper stabilizer chain and we have finished processing of the new generator a.

We now describe this procedure in a formal way. In the version presented here, the algorithm picks base points itself, though one can obviously "seed" a partial base.

[1] The conditions are trivially all necessary.

ALGORITHM II.8: Recursive version of the Schreier-Sims algorithm. As the main algorithm is a recursive function (EXTEND), we need to perform a separate initialization.

Input: A generating set $\underline{\boldsymbol{g}}$ for a permutation group G
Output: A recursive data structure for a stabilizer chain for G.

begin
1: C := rec(generators := [ ]);
2: **for** $a \in \underline{\boldsymbol{g}}$ **do**
3: EXTEND(C, a);
4: **od**;
5: **return** C;
end

The actual work is then done in the following recursive function which extends and modifies the (full or layer) chain C.

EXTEND(C, a)

begin
1: **if** ElementTest(C, a) fails **then** {Extend existing stabilizer chain}
2: **if** C.generators = [ ] **then** {We are on the bottom of the chain}
3: C.stabilizer := rec(generators := [ ]); {Add a new layer}
4: β := one point moved by a; {or a predefined next base point}
5: Add a to C.generators;
6: C.orbit := $[\beta]$;C.transversal := [1];
7: $\delta := \beta^a$; $s := a$;{Special orbit algorithm for single generator}
8: **while** $\delta \neq \beta$ **do**
9: Add δ to C.orbit; Add s to C.transversal;
10: $\delta := \delta^a$; $s := s \cdot a$;
11: **od**;
12: EXTEND(C.stabilizer, s);{s is only Schreier generator}
13: **else** {The layer already has an existing orbit}
14: $O := C$.orbit; $T := C$.transversal; {Extend orbit algorithm}
15: $l := |O|$;
16: **for** $\delta \in O$ in position 1 to l **do** {Old points only with new generator}
17: $\gamma = \delta^a$;
18: **if** $\gamma \notin O$ **then**
19: Add γ to O; update transversal;
20: **else**
21: $s := T[\delta]aT[\gamma]^{-1}$;
22: EXTEND(C.stabilizer, s);
23: **fi**;
24: **od**;
25: **for** $\delta \in O$ in position $> l$ **do** {new points with all generators}
26: **for** $b \in C$.generators $\cup \{a\}$ **do**
27: $\gamma = \delta^b$;

```
28:            if γ ∉ O then
29:               Add γ to O; update transversal;
30:            else
31:               s := T[δ]bT[γ]^{-1};
32:               EXTEND(C.stabilizer, s);
33:            fi;
34:         od;
35:      od;
36:      Add a to C.generators;
37:   fi;
38: fi;
end
```

PERFORMANCE II.9: We only process a if the element test in line 1 fails. In this case the test will fail on some (potentially lower) layer in the chain after already dividing off transversal elements on a higher layer. As this "sifted" element differs from a by existing transversal factors it clearly does not change the reulting group. However as it is moving fewer points, it is preferably taken in place of a. This way it will give immediately Schreier generators on a lower layer.

NOTE II.10: One can show (see [Ser03]) that the resulting algorithm has a complexity which polynomial in the degree n.

NOTE II.11: If G is known to be solvable, there is a better algorithm that has been proposed by Sims in 1990 [Sim90].

Strong Generators

The reason for the recursive structure of the Schreier-Sims algorithm is that we do not know immediately a reasonable set of generators for the different stabilizers. If we did, we could build the stabilizer chain very quickly layer by layer, just using the orbit algorithm. This motivates the following definition:

DEFINITION II.12: A *Strong generating system* (SGS) for G is a generating set S for G, such that the i-th stabilizer $G^{(i)}$ is generated by $S \cap G^{(i)}$.

If we have a stabilizer chain, the union of the `generators` components on all layers obviously yields a strong generating system.

Given a strong generating set, we can thus very easily rebuild a stabilizer chain. This explains, why the computation of a stabilizer chain is often described as computation of a base and a strong generating system.

PERFORMANCE II.13: A small problem in the construction of a Schreier vector is that the algorithm may produce unwieldy large expressions for representatives. Consider for example the group

$$G = \langle a = (1,2,3,4,\ldots,100), b = (1,2)\rangle\,.$$

With this generating set, the representative for i will be a^i, respectively for $i > 50$ the power $a^{-(101-i)}$. On the other hand, as $G = S_{100}$, there are other generating sets, which produce in average much shorter representative words.

This problem is magnified by the fact that we iteratively add generators.

A way around this problem is to add further group elements (short products of the existing generators) to the generating set.

In particular, one could rebuild the stabilizer chain with a strong generating set as new generators to obtain immediately multiple generators on each layer.

Base images and Permutation words

The most expensive subtask of the Schreier-Sims algorithm is the multiplication of permutations, in particular if we have to get transversal elements from a Schreier vector. To improve performance, it is thus desirable to reduce the number of multiplications.

There are two approaches to do this:

Base Images: If we already know a base $B = (\beta_1,\ldots,\beta_m)$, we know that every permutation g is determined uniquely by the *base image* $(\beta_1^g,\ldots,\beta_m^g)$ it produces.

Now suppose that $(\gamma_1,\ldots,\gamma_m)$ is a base image under some group element g and we have $h \in G$. Then $(\gamma_1^h,\ldots,\gamma_m^h)$ is the base image for gh.

We thus can represent group elements in the algorithm by their base images. The cost of one multiplication then is proportional to the length of a base and not, as permutation multiplication would be, to the length of the domain.

This is in particular relevant if we work with a subgroup $U \le G$ and have already a base for G determined.

Words: Instead of multiplying out permutations, we can store products as a *word* of permutations, i.e. fgh is stored as $[f,g,h]$. Multiplication of words is simple concatenation; the inverse of $[f,g,h]$ is $[h^{-1},g^{-1},f^{-1}]$; the image of a point ω under $[f,g,h]$ can be computed as $((\omega^f)^g)^h$. The only test which is hard, is to determine whether a word represents the identity. For this we need to compute the images of **all** points (unless we know a base).

Randomization

The biggest problem with the Schreier-Sims algorithm is the large number of Schreier generators on each layer – the problem is that we have no criterion which elements we can safely ignore.

Experiments show that one can usually ignore at least half the generators, but there are more problematic cases. This can be rectified, to give a statistically satisfactory behavior, but is a rather complicated process.

Another way to look at this is that if we only pick some Schreier generators, we effectively rebuild a stabilizer chain with a set S which claims to be a strong generating set, but is in effect a proper subset. Consequentially the resulting partial chain is not describing the group but a proper subset. As every proper subgroup

has index 2 one would thus expect that the element test with this chain will fail with probability $\frac{1}{2}$ for a random group element. Indeed this is true as the following lemma shows:

LEMMA II.14: Suppose we have built a partial stabilizer chain for a group G which is missing Schreier generators on some layers (and thus — by lemma II.7 — has too short orbits on some layers). Then an element of G fails the element test for this chain with probability at least $\frac{1}{2}$.

Proof: Let $S^{(j)}$ be the groups generated by the Schreier generators on the respective layer of the chain and $G^{(j)}$ the correct stabilizers. Let i be the largest index in the stabilizer chain, such that $S^{(i+1)} \neq \mathrm{Stab}_{S^{(i)}}(\beta_i)$. Then $S^{(i+1)}$ in fact has a proper chain (otherwise i was larger) and we can do a true element test in this group.

Now consider the element test for group elements with the given chain S. Suppose that the probability is p, that a uniformly random element $g \in G$ sifts through layer 1 to i. Let $\tilde{g}$ be the product of transversal elements divided off at this point. Then $r = g/\tilde{g} \in G^{(i+1)}$. Furthermore (multiply one element that passes with elements of $G^{(i+1)}$) every element of $G^{(i+1)}$ occurs as remainder r for a random g with the same probability.

On the other hand, by the choice of i, we know that $S^{(i+1)} \neq G^{(i+1)}$, thus $\left[G^{(i+1)}:S^{(i+1)}\right] \geq 2$. Thus r passes the element test for $S^{(i+1)}$ with probability $\leq \frac{1}{2}$. Sifting thus fails at least with probability

$$(1-p) + p\frac{1}{2} = 1 - \frac{p}{2} \geq \frac{1}{2}$$

□

If we suppose that generators passed to the next layer of the Schreier-Sims algorithm are uniformly distributed (which is not true, but often not too wrong), we can thus take the passing of the element test to indicate with probability $\geq \frac{1}{2}$ that the chain is in fact correct. If subsequent Schreier generators do not extend the chain, this probability grows. One thus could stop processing further Schreier generators, once a fixed number of Schreier generators in a row did not extend the chain.

Furthermore, in the "Random Schreier-Sims" algorithm as proposed in [Leo80], we can form (Pseudo-)random elements of the group G (e.g. using algorithm I.21) and test whether a fixed number (20 elements is used in [Leo80]) of them pass the element test with the existing chain.

Verification

The "only" problem with even the best randomized approach is that we can never guarantee that we obtained a correct stabilizer chain. If[2] we want to obtain proven results, we need to *verify* the obtained chain.

[2] a rhetorical "if" as a mathematician

The following methods can be used for such a verification:

Known Order By lemma II.7 a partial chain will not be proper if the orbit on some layer becomes to short. In this situation the order of the group as calculated from the stabilizer chain is too small. If we know $|G|$ we can simply compare.

Combinatorial verification Charles Sims developed in 1970 an combinatorial algorithm for verifying a stabilizer chain obtained with random methods but did not publish the method. The first description can be found in [Ser03].

Presentations One can use the stabilizer chain to deduce "relations" which have to hold among the generators of the group – if the chain is too small some will fail. This will lead to a so-called Todd-Coxeter-Schreier-Sims algorithm, see section III.10.

Using a Composition series If we know a composition series, we can verify all composition factors separately, see III.10

If the verification of a chain fails, we have to continue adding Schreier generators. (Often the failure of a test already provides a particular element that should be used.)

Changing the base

In some situations it is desirable to have a stabilizer chain for a particular base. We can certainly achieve this by building a new stabilizer chain. If a chain already exists, we know the order of the group, and thus can safely use a randomized approach.

Still in many cases when we want to change only a few points in an existing base this is too expensive. In such a situation it merits to modify an existing base. Let us assume that we know a base $B = (\beta_1, \ldots, \beta_m)$.

The easy case is if the new base is in fact a possible base image for G, i.e. the new base is $(\beta_1^g, \ldots, \beta_m^g)$ for $g \in G$. (Such an element g can be found easily, if it exists, using the stabilizer chain!)

In this situation, we can simply *conjugate* the whole stabilizer chain (i.e. conjugate all generators by g, take the image of all points under g) and obtain the desired chain.

In general (unless the group is the symmetric group), the new base $\Gamma = (\gamma_1, \ldots, \gamma_n)$ will not be a base image. In this situation we first try, using the base image approach, to move some base points in B to points in Γ, preferably at the same position, but even different positions are fine. Call this new base E.

Then we add the remaining points of Γ to E, by introducing trivial stabilizer steps (i.e. we have orbit 1 and all generators are Schreier generators). This is certainly possible on some layer of the chain, but it might be the bottom layer. The resulting base is called H.

Next we use a *base swap* procedure (see [HEO05, 4.4.7]) that will exchange the order of two subsequent base points η_i and η_j in H. (We only need to modify two

subsequent entries in the stabilizer chain, as the previous and following stabilizers will be equal.)

Using this procedure, we move the base points in Γ (in the right order) to the start. Finally we delete trivial stabilizer steps at the end of the chain.

II.2 Consequences of Schreier-Sims

Using a stabilizer chain we can perform a variety of calculations for a group G:

- Test whether a permutation $g \in G$
- Given a base image $[\gamma_1, \ldots, \gamma_m]$ find, if possible, an element $g \in G$, such that $\beta_i^g = \gamma_i$: This is really just a modified element test in which we use the transversal elements corresponding to the base images.
- Calculate $|G| = \left|\beta_1^G\right| \cdot \left|G^{(1)}\right| = \left|\beta_1^G\right| \cdot \left|\beta_2^{G^{(1)}}\right| \left|G^{(2)}\right| = \cdots$ as the product or the orbit lengths.
- Normal Closure with proper element test.
- Determine the sizes of groups in the derived series $D_0 = G, D_i = D'_{i-1}$ and lower central series $L_0 = G, L_i = [G, L_{i-1}]$.
- Determine whether G is solvable or nilpotent.
- Test whether two elements are in the same coset of a subgroup.
- Determine the permutation action on the cosets of a subgroup.
- Determine the point wise stabilizer of a set (i.e. the subgroup stabilizing all points in the set) by calculating a stabilizer chain for a base starting with the points from the set.
- Enumerate G, i.e. assign to every element a number and have efficient functions to translate element to number and vice versa: We noted already that we can easily translate between elements and base images. We consider each base image as a list of numbers, according to the position of the point in the orbit. This is the "multi-adic" representation of a number $\in \{1, \ldots, |G|\}$.
- Obtain random elements with guaranteed equal distribution.

Factorization and Homomorphisms

We have noted before that the element test algorithm II.5 will express a group element g as a product of transversal elements. On the other hand, every transversal element has been obtained as a product of the generators. By keeping track of how these transversal elements arose as products of the *original* generators, we can thus express any group element as a word in the generators.

Note II.15: This looks like a perfect functionality for solving puzzles, such as Rubik's Cube. Alas the words obtained are *horribly* long and in practice infeasible. One way used to obtain short words [Min98] is to add many short words in the original generators to the original generating set, thus automatically obtaining shorter words for stabilizer generators on lower layers. Explicit bounds have been proven recently [Rok08] using enormous[3] calculations.

A main use of this is in implementing homomorphisms. Suppose that G is a permutation group and we have a homomorphism $\varphi: G \to H$ given by a generating set $\underline{\boldsymbol{g}}$ of G and the images $\underline{\boldsymbol{g}}^\varphi$.

Then expressing an element $x \in G$ as word in $\underline{\boldsymbol{g}}$ lets us evaluate the same word in $\underline{\boldsymbol{g}}^\varphi$, which must be the image x^φ.

To speed up the way products of the generator images are formed, we also store images for the Schreier generators – this way comparatively few products have to be evaluated. We obtain these images, by building a *new* stabilizer chain for G that is only used for the homomorphism. (As we can assume that $|G|$ is known, we can use a random Schreier-Sims algorithm with easy verification.)

The elements for which this chain are formed however are not elements of G, but elements of $G \times H$. We consider only the G-part for purposes of building the stabilizer chain, the H part then just mirrors the multiplication.

The calculation then starts with a generating set of the form $\{(g, g^\varphi) \mid g \in \underline{\boldsymbol{g}}\}$.

Kernel: If H is also a permutation group, we can represent the direct product as a permutation group by moving the points on which H acts, i.e. for $S_3 \times S_4$ the element $((1,2),(3,4))$ is represented by $(1,2)(6,7)$. The domain Ω then decomposes in $\Omega_G \cup \Omega_H$.

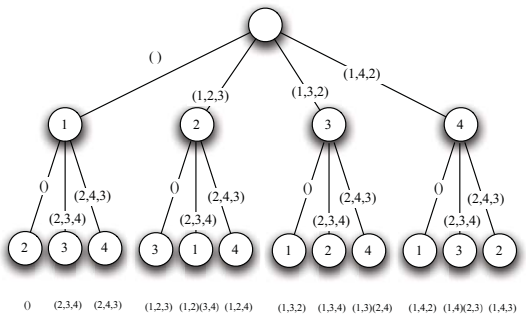
Let $D = \left\langle (g, g^\varphi) \mid g \in \underline{\boldsymbol{g}} \right\rangle$ the group (the "diagonal" subgroup of the direct product) representing the homomorphism φ. Then the point wise stabilizer $\mathrm{Stab}_D(\Omega_H)$ corresponds to the set of elements whose image is trivial, i,e, its G-projection is the kernel of φ.

II.3 Backtrack

By "backtrack" we mean an algorithm that will – by traversing a tree fromed from a stabilizer chain – run (in worst case) through all elements of a permutation group. It will find (one or all) elements fulfilling a certain property. The input being generators of a subgroup of S_n (so in an extreme case 2 permutations of degree n generating a group of order $n!$) such an algorithm has runtime exponential in its input size. However is so far the best method known[4] for tasks such as

- Centralizer and Normalizer in permutation groups
- Conjugating element in permutation groups

[3] Calculations were done using the spare cycles of the rendering farm of a Hollywood studio!
[4] and better methods might impact the question of whether P=NP



Vertices are the images for the base point 1 and 2 respectively. Edge labels are the transversal elements. The permutations under the leafs are the resulting group elements.

Figure II.1: Tree structure for A_4

- Set stabilizer and set transporter
- Graph isomorphism

Basic backtrack

The basic version of backtrack takes a permutation group G and builds a tree from a stabilizer chain of G: The levels of the tree correspond to the layers of the stabilizer chain. Each node corresponds to a (partial) base image $(\beta_1^g, \ldots, \beta_k^g)$ $(k \leq m)$. The branches down from such a node then correspond to the orbit of β_{k+1} under $G^{(k)}$. Since a partial base image for the preceding points is already prescribed, the branches are labelled not with the orbit $orb := \beta_{k+1}^{G^{(k)}}$, but with the images of orb under an element g yielding the partial base image[5].

Figure II.1 shows this enumeration for the example of $G = A_4$.

Again, as we consider stabilizer chains as recursive objects, this is a recursive algorithm.

Input: We are passing a (sub)chain (which describes the tree structure below) C and a partial product of representatives r, that describes the tree node.
Output: The program prints out all group elements
begin
1: $leaf$:= |C.`stabilizer.generators`| = 0; {have we reached a leaf of the tree?}
2: Δ := C.`orbit`;

[5] The choice of g does not impact the set of images

3: **for** $\delta \in \Delta$ **do**
4: x := C.`transversal`$[\delta]$;
5: **if** $leaf$ **then**
6: Print $x \cdot r$;
7: **else**
8: Call recursively for C.`stabilizer`, $x \cdot r$;
9: **fi**;
10: **od**;
end

We start with the whole chain for G and offset $r = ()$.

Obviously, instead of printing the elements, we can test the elements for whatever property we desire and collect the elements which yield a correct answer.

In this version we are running always in the same way through the orbit. For several practical (see below) and aesthetic reasons, it can be desirable to run through elements in a lexicographically ordered way (i.e. compare permutations as base images for the base $\{1, 2, 3, \ldots\}$). Then the possible images of the base point are given by the orbit points (that's what we chose) mapped under r (as we post-multiply by r).

We can achieve this by sorting Δ in line 2 according to the images under r, in GAP notation `SortParallel`$(\{\delta^g \mid \delta \in \Delta\}, \Delta)$.

Pruning

The problem of the basic backtrack routine is that running through all elements of a larger group will be rather time intensive. A principal aim for any backtrack search is therefore to prune the search tree.

This pruning is possible if we are searching only for elements fulfilling a particular property: It is possible that a partial base image already eliminates all elements which have these base point images as candidates for satisfying the property.

Example II.16: As an example of such a test, suppose we are looking for an element that maps $(1,2)(3,4,5)$ to $(2,4)(1,5,3)$. We chose a base starting with $\{1,2\}$.

As an n-cycle must be mapped to an n-cycle, the image of 1 can be only 2 or 4, eliminating all top branches but two. Furthermore, if $1^g = 2$, we know that $2^g = 4$; respectively $1^g = 4$ implies $2^g = 2$. On the second layer we thus have but one branch.

Similar restrictions will hold for the subsequent base points.

An improved backtrack algorithm therefore will, every time a new base image is selected, employ a (problem-dependent!) test, whether group elements with this partial base image can in fact fulfill the desired property. Only if they can, lines 5-9 are executed.

Example II.17: We want to find the centralizer of $(1,2,4)(5,6,8)$ in $G = \langle (1,3,5,7)(2,$
This group has order 24, we pick base $(1,2)$ and get the chain:

```
rec( generators := [ (1,3,5,7)(2,4,6,8), (1,3,8)(4,5,7) ],
```

```
orbit := [ 1, 3, 5, 8, 7, 2, 4, 6 ],
transversal := [ (), (1,2,7,5,6,3)(4,8), (1,3,5,7)(2,4,6,8),
    (1,4,2)(5,8,6), (1,5)(2,6)(3,7)(4,8), (1,6,7)(2,3,5),
    (1,7,5,3)(2,8,6,4), (1,8,2,5,4,6)(3,7) ],
stabilizer := rec( generators := [ (2,8,7)(3,6,4) ],
    orbit := [ 2, 8, 7 ],
    transversal := [ , (),,,,, (2,7,8)(3,4,6), (2,8,7)(3,6,4) ]
    stabilizer := rec( generators := [  ] ) ) )
```

We can map 1 to 1, 2, 4, 5, 6, 8. In each case the image of 2 is then fully determined:

1^g	2^g	x	Works?
1	2	()	$\surd$
2	4	(1,2,4)(5,6,8)	$\surd$
4	1	(1,4,2)(5,8,6)	$\surd$
5	6	(1,5)(2,6)(3,7)(4,8)	$\surd$
6	8	(1,6,4,5,2,8)(3,7)	$\surd$
8	5	(1,8,2,5,4,6)(3,7)	$\surd$

At this point we have actually found *all* elements in the centralizer.

Such pruning conditions obviously are problem specific. When intelligently applied, they can often eliminate large parts of the search space. This usually also requires a suitable choice of base.

Example II.18: Suppose we want to find the *setwise* stabilizer of $\Delta \subset \Omega$[6]. We choose a base whose initial points are chosen from within Δ as far as possible, say $\beta_1, \ldots, \beta_k \in \Delta$ and $G^{(k)}$ moves no point in Δ. Then clearly $G^{(k)} \leq \mathrm{Stab}_G(\Delta)$. Furthermore the possible images for β_i $(i \leq k)$ are restricted to Δ.

Example II.19: We want to find an element g conjugating the permutation x to y. A first, easily tested, necessary condition is that the cycle structure of x and y is the same; we now assume that this is the case. We now chose the first base point β_1 within a long cycle of x whose length l occurs rarely (so there are few cycles in y of this length). Then β_1 must be mapped to a point which in y occurs in a cycle of length l if the element g is to map x to y. Furthermore $x^g = y$ if and only if $x^{gy} = y$. We therefore need to consider only one possible image per cycle in y of the correct length. Subsequent base points then are chosen from the same cycle in x. For any such base point $\beta_1^{x^k}$ the image under g must be $(\beta_1^{x^k})^g = (\beta_1^g)^{y^k}$, i.e. it is uniquely determined bythe choice of β_1^g.

In the following discussion we will assume that we have chosen a suitable base, and that we are doing such problem-specific pruning.

[6] Without loss of generality, assume that $|\Delta| \leq \frac{|\Omega|}{2}$, otherwise we consider the complement $\Omega - \Delta$

Note II.20: Newer version of backtrack algorithms, so called "Partition backtrack" routines label the tree not with base images, but with ordered[7] partitions of Ω. The partial base image $(\gamma_1, \ldots, \gamma_k)$ then corresponds to a partition with each γ_i $(i \leq k)$ is in its own cell, a leaf of the tree corresponds to a partition with all points in a cell of their own.

So far this is just a different description of the basic backtrack algorithm. A difference is seen, however, once one searches for elements with particular properties. The condition to stabilize points (or map points in a certain way) can impose conditions on other points (and consequentialy split the remaining cell). For example when centralizing $(1,2,3)(4,5,6)$ if we stabilize 1 we also have to stabilize 4.

One can describe such conditions by intersecting the backtrack partition with a propert-depending partition.

The effect of this is that the tree of the backtrack search becomes more shallow.

Properties defining subgroups

For most properties interesting in a group-theoretic context, the set of elements fulfilling the condition we search for actually forms a subgroup, respectively a double coset. (A *double coset* is a subset of elements of the form $SgT = \{sgt \mid s \in S, t \in T\}$ for $S, T \leq G$.) For example:

- Centralizer, Normalizer, set stabilizer, automorphism group of a graph are subgroups.
- In a conjugacy test: Find g with $a^g = b$. Here the fulfilling elements are in a double coset $C_G(a) \cdot h \cdot C_G(b)$ if h is one solution.
- Testing for isomorphism between the graphs Γ and Θ. If h is one isomorphism, the set of isomorphisms has the form $\mathrm{Aut}(\Gamma)h\,\mathrm{Aut}(\Theta)$.

We will now consider only the case of a subgroup, the double coset case is similar. We want to find all elements in G that fulfill a testable property. We assume that this set of elements forms a subgroup $P \leq G$.

Clearly we only need to find a generating set of P (and chances are good that a few random elements of P will generate P). Therefore much time will be spent in proving that no element *outside* the subgroup we found so far fulfills the property. So let us suppose we know a subgroup $K \leq P$ (which might be trivial). Also whenever we find a new element $g \in P$, we update $K := \langle K, g \rangle$.

Note II.21: When testing for a single "mapping" element (e.g. in a conjugacy test) of course we are not deliberately building such a subgroup K. However we can still do so (essentially for free) if we *happen* to come upon an element stabilizing the initial object. This way similar benefits are obtained.

[7] I.e. the order in which the cells occur is relevant

Our strategy will be "left-first", i.e. we first enter the stabilizer of a base point, before considering any coset. Thus we will have examined the whole of $G^{(i)}$ before considering any other elements of $G^{(i-1)}$. In particular, we can assume that we know $G^{(i)} \cap P$ before testing any element of G outside $G^{(i)}$.

Note II.22: This observation also shows that the backtrack search will automatically produce a strong generating set for P (or the subgroup K of elements found so far). We can thus assume (at little cost) that we have a stabilizer chain for K (and that the algorithm will return a stabilizer chain of P).

If we make this assumption, we can decribe criteria for pruning the search tree:

Lemma II.23: Suppose we know $K = G^{(l)} \cap P$ and that $\mathcal{N}$ is a node which prescribes the first l base images. Suppose we find an element g below $\mathcal{N}$ that is in P. Then we can discard the whole remaining subtree below $\mathcal{N}$.

Proof: Any further element of P in this subtree is in the coset Kg. □

This test works if we find new elements, but we can do much better: Suppose we test an element $g \notin K$. Then either $g \in P$, in which case we increase K by at least a factor 2. Or $g \notin P$, but then no element in the double coset KgK can be in P either.

While this condition has the potential to reduce the search space enormously (making the cost more proportional to $|P\backslash G/P|$ than to $|G|$), the problem is just how to incorporate it in the backtrack search.

What we would like to do is to test every double coset KgK only once. A standard method for such duplicate rejection (without explicitly storing all elements of KgK for every g tested) is to define a "canonical" representative for each double coset. Then every element g that is not canonical for its double coset can be discarded (as we will test the – different – canonical representative at another time).

Typically the definition of "canonical" will require some arbitrary symmetry-breaking condition (all elements are images under a group, so they are in some way "the same"). What we will use is that the element is minimal with respect to a comparison of base images (i.e. we lexicographically compare the base images $(\beta_1^g, \beta_2^g, \ldots)$) among all elements in the double coset. Note that by sorting the orbits the basic backtrack algorithm will run through elements in this ordering.

Unfortunately finding the smallest element in a double coset (or testing whether one element is smallest) is hard. We will instead use weaker conditions, that adapt well to the tree traversal strategy.

The first condition does in fact only use minimality in the left coset gK:

Lemma II.24: Suppose that $\mathcal{N}$ is a node in the search tree that prescribes the first l base images as $(\gamma_1, \ldots, \gamma_l)$ and that $K \leq P$ is the subgroup found so far. If g lies under $\mathcal{N}$ and is the smallest element of KgK then γ_l is minimal in the orbit $\gamma_l^{\mathrm{Stab}_K(\gamma_1,\ldots,\gamma_{l-1})}$.

Proof: Suppose not. Let $h \in \mathrm{Stab}_K(\gamma_1, \ldots, \gamma_{l-1})$ such that $\gamma_l^h < \gamma_l$. Then $gh \in KgK$ and $gh < g$, contradiction. □

To use this lemma we need to perform a base change to find the stabilizer $Stab_K(\gamma_1, \ldots, \gamma_{l-1})$. Note that we will already know $Stab_K(\gamma_1, \ldots, \gamma_{l-2})$, so little extra work is needed.

The next criterion eliminates certain base images from consideration.

Lemma II.25: Suppose that $\mathcal{N}$ is a node in the search tree that prescribes the first l base images as $(\gamma_1, \ldots, \gamma_l)$ and that $K \leq P$ is the subgroup found so far. Let $R := Stab_G(\gamma, \ldots, \gamma_{l-1})$, $S := Stab_K(\beta_1, \ldots, \beta_{l-1})$, and $s = \left|\beta_l^S\right|$.

If g lies under $\mathcal{N}$ and is the smallest element of KgK then γ_l cannot be among the last $s-1$ elements of its orbit under R.

Proof: Let $\Gamma = \{\beta_l^{hg} \mid h \in S\} = (\beta_l^S)^g$. Then $|\Gamma| = s$ and $\gamma_l = \beta_l^g \in \Gamma$.

As any product $hg \in Kg \subset KgK$, the minimality of g implies that $\gamma_l = \min \Gamma$.

If $\gamma = \beta_l^{hg} \in \Gamma$, then $\gamma^{g^{-1}h^{-1}g} = \gamma_l$ with $g^{-1}h^{-1}g \in R = (G^{(l-1)})^g$. Thus $\Gamma \subset \gamma_l^R$ and γ_l^R must contain at least $s-1$ elements larger than γ_l. □

More details (and further criteria) can be found in [Ser03].

II.4 Natural Actions and Decompositions

The algorithms we have seen so far in this chapter were mainly combinatorial in nature and uses only a small amount of group theory. We now want to look at the computation of more structural information, for example a composition series. (Later we will (see III.10) that such calculations actually are the key towards efficient stabilizer chain computations.)

The fundamental idea will be to take a given permutation group $G \leq S_n$ and to split it apart into a normal subgroup $N \lhd G$ and a factor group G/N, again represented as permutation groups, by finding a suitable action which gives a homomorphism $\varphi: G \to S_m$ with $N = \mathrm{Kern}\,\varphi$.

In this section we will be looking at actions that arise from the natural permutation action. We shall describe these actions, and show how a permutation group relates to the images of these actions. Much of this is theory that is of interest on its own. More details can be found in books on permutation groups such as [DM96] or [Cam99].

We will be talking about permutation groups. If G is a group with a *permutation action* φ on Ω, the corresponding statements remain true if we interpret them for the factor $G/\mathrm{Kern}\,\varphi$.

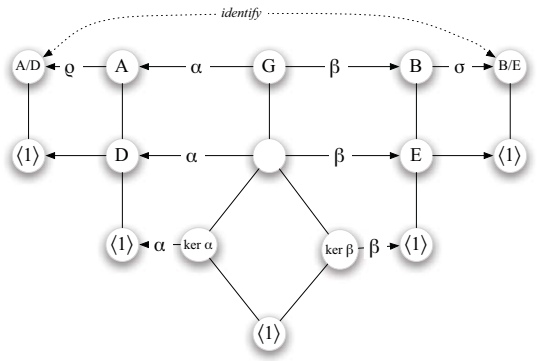


Figure II.2: Subdirect Product

Orbits: Intransitive Groups

The first situation we want to look at is that of an intransitive group, i.e. a permutation group which has multiple orbits on its permutation domain:

Suppose we have that $\Omega = \Delta \uplus \Gamma$ and both Γ and Δ are orbits[8]. In this situation we get two homomorphisms, $\alpha\colon G \to S_\Gamma$ and $\beta\colon G \to S_\Delta$, such that $\operatorname{Kern}\alpha \cap \operatorname{Kern}\beta = \langle 1\rangle$. We set $A = G^\alpha$ and $B = G^\beta$

Now form a new homomorphism, $\epsilon\colon G \to A \times B$, defined by $g^\epsilon = (g^\alpha, g^\beta)$. Then $\operatorname{Kern}\epsilon = \operatorname{Kern}\alpha \cap \operatorname{Kern}\beta = \langle 1\rangle$.

We can thus consider G as (isomorphic to) a subgroup of $A \times B$, which will project on both components with full image. Such a group is called a *subdirect product*, the construction is due to Remak [Rem30].

(We do not really need that G is a permutation group, we just have two homomorphisms, whose kernels intersect trivially; respectively two normal subgroups which intersect trivially.)

We now want to make this construction synthetic, i.e. we want to describe $\operatorname{Image}(\epsilon)$ in terms of A and B.

For this we set $D = (\operatorname{Kern}\beta)^\alpha \lhd A$ and $E = (\operatorname{Kern}\alpha)^\beta \lhd B$. Then (isomorphism theorem!)

$$A/D = G^\alpha/(\operatorname{Kern}\beta)^\alpha \cong G/\langle \operatorname{Kern}\alpha, \operatorname{Kern}\beta\rangle \cong G^\beta/(\operatorname{Kern}\alpha)^\beta = B/E,$$

i.e. we have isomorphic factor groups of A and B. See figure II.2.

Let $\rho\colon A \to A/D$ and $\sigma\colon B \to B/E$ the natural homomorphisms and $\zeta\colon A/D \to B/E$ the isomorphism given by $(g^\alpha)^\rho \mapsto (g^\beta)^\sigma$. We therefore have for the elements

[8] or unions of orbits. We do not need that the action on Γ and Δ is transitive.

of G^ϵ, that

$$G^\epsilon = \left\{(a,b) \in A \times B \mid (a^\rho)^\zeta = b^\sigma\right\}.$$

We now use this identity for the synthetic construction (the "external" subdirect product): Suppose we have two groups Assume that $\zeta\colon A/D \to B/E$ is an isomorphism. The set

$$A \perp B = \left\{(a,b) \in A \times B \mid (a^\rho)^\zeta = b^\sigma\right\} \leq A \times B$$

is called the subdirect product of A and B. It is an easy exercise to see that $A \perp B$ is a group and that its image under the projections from $A \times B$ onto A and B is the full component.

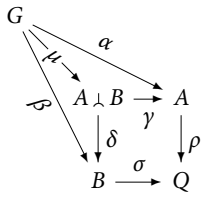
Note II.26: The notation $A \perp B$ is misleading: the product also depends on the choice of factor groups as well as on ζ. We can say that it is the subdirect product in which the factor groups A/D and B/E are "glued together".

Note II.27: If we consider $A \times B$ as a permutation group acting on $\Delta \uplus \Gamma$, then $A \perp B$ arises naturally as an intransitive group, by labelling the points consistently, we get that $G = G^\epsilon$ as permutation groups.

Note II.28: Instead of identifying two factor groups explicitly via the isomorphism ζ, one also could simply consider one group Q together with epimorphisms $\rho\colon A \to Q$ and $\sigma\colon B \to Q$. In this context the subdirect product is also sometimes denoted by $A \times_Q B$.

The following property describes the subdirect product in a categorial context as the *fibre product* (or *pullback*) in the category of groups:

Lemma II.29: Let A, B, Q be groups and $\rho\colon A \to Q$ and $\sigma\colon B \to Q$ both epimorphisms[9]. We consider the subdirect product $A \perp B$ with respect to these homomorphisms. Let G be a group which makes the diagram $\begin{array}{ccc} G & \xrightarrow{\alpha} & A \\ {\scriptstyle\beta}\downarrow & & \downarrow{\scriptstyle\rho} \\ B & \xrightarrow{\sigma} & Q \end{array}$ commutative (I.e.: There exist homomorphisms $\alpha\colon G \to A$ and $\beta\colon G \to B$ such that for every $g \in G$ we have that $g^{\alpha\rho} = g^{\beta\sigma}$). Then there exists a unique map $\mu\colon G \to A \perp B$, such that the diagram



[9] One can drop the condition that ρ and σ have to be surjective by considering subgroups of A and B instead.

(with γ, δ the obvious projections of the subdirect product) is commutative.

Proof: If G makes the diagram commutative, then G is a subdirect product of $G^\alpha \leq A$ with $G^\beta \leq B$ and as such embeds into $A \times B$ as $G^\epsilon = \left\{(g^\alpha, g^\beta) \in A \times B\right\}$. We observe (commutativity of the diagram!) that $g^{\alpha\rho} = g^{\beta\sigma}$. Therefore

$$G^\epsilon \leq \{(a,b) \in A \times B \mid (a^\rho) = b^\sigma\} = A \curlywedge B$$

We now set μ to be the corestriction[10] of ϵ to $A \curlywedge B$. Clearly ($\mu\gamma = \epsilon\gamma$ is the projection of G onto its A-part and similarly for B) this makes the diagram commutative.

To show the uniqueness of μ note that the conditions $\mu\gamma = \alpha$ and $\mu\delta = \beta$ already prescribe the image $g^\mu \in A \curlywedge B$. □

Returning to the situation of a permutation group, we see that every group with two sets of orbits is a subdirect product of two groups of smaller degree. Thus every permutation group is obtained by forming iteratively subdirect products of transitive groups. (One could try to define the product of more than 2 factors, in practice it is far easier to simply consider the iterative construction.)

To classify all permutation groups of a given degree n we thus would need to:

- Classify all transitive groups up to this degree.
- Form their iterated subdirect products (such that the degrees sum up to $\leq n$).

Blocks: Imprimitive Groups

> Permutations par groupes croissant de lettres:
> Rvers unjou urlap midis ormea latef eduna
>
> Exercices de style
> Raymond Queneau

Let us now consider a group G acting transitively on Ω.

Definition II.30: A partition $\mathcal{B} = \{B_1, \ldots, B_k\}$ of Ω (I.e. we have that $B_i \subset \Omega$ and Ω is the disjoint union of the B_i) is a *block system*, if it is invariant under G. I.e. the set-wise image $B_i^g \in \mathcal{B}$ for every $g \in G$. We call the subsets B_i the *blocks*.

Note II.31: The following two block systems always exist. They are called the *trivial block systems*:

$$\mathcal{B}_1 = \{\{\omega\} \mid \omega \in \Omega\}, \qquad \mathcal{B}_\infty = \{\Omega\}$$

Definition II.32: A group is acting *imprimitively* on Ω if G acts transitively, and affords a nontrivial block system. Otherwise we say the group acts *primitively*.

[10] The function defined by the same rule, but a restricted range

Lemma II.33: Let $\mathcal{B} = \{B_1, \ldots, B_k\}$. Then for every i, j there exists $g \in G$, such that $B_i^g = B_j$. In particular $|B_i| = |B_j|$ and thus $|\Omega| = |B_1| \cdot |\mathcal{B}|$.

Proof: Let $\delta \in B_i$ and $\gamma \in B_j$. As G acts transitively, there is $g \in G$ such that $\delta^g = \gamma$. Thus $B_i^g \cap B_j \neq \varnothing$. As the partition is kept invariant we have that $B_i^g = B_j$. □

Corollary II.34: A block system is determined by one block – the other blocks are just images.

Corollary II.35: Any transitive group of prime degree is primitive.

The following lemma explains the group-theoretic relevance of block systems:

Lemma II.36: Suppose G acts transitively on Ω and let $S = \mathrm{Stab}_G(\omega)$ for some $\omega \in \Omega$. Then there is a bijection between subgroups $S \leq T \leq G$ and block systems $\mathcal{B} = \{B_1, \ldots, B_k\}$ for G on Ω.

Using the convention that B_1 is the block containing ω, the bijection is given by $T = \mathrm{Stab}_G(B_1)$, respectively by $B_1 = \omega^T$.

Proof: Suppose that $S \leq T \leq G$. We set $B = \omega^T$ and $\mathcal{B} = B^G$ and claim that $\mathcal{B}$ is a block system:

Let $g, h \in G$, and suppose that $B^g \cap B^h \neq \varnothing$. Then there exists $\delta, \gamma \in B$ such that $\delta^g = \gamma^h$. As $B = \omega^T$ we have that $\delta = \omega^s$, $\gamma = \omega^t$ for $s, t \in T$. Thus $\omega^{sg} = \omega^{th}$, and thus $sgh^{-1}t^{-1} \in \mathrm{Stab}_G(\omega) = S \leq T$. This implies that $gh^{-1} \in T$. As T stabilizes B (by definition), we thus have that $B^g = B^h$. Thus the images of B under G form a partition of Ω. Because it was obtained as an orbit, this partition is clearly G-invariant.

Vice versa, let $\mathcal{B}$ be a block system and let $\omega \in B \in \mathcal{B}$ be the block containing ω. Then any $g \in G$ which fixes ω has to fix B, thus $\mathrm{Stab}_G(\omega) \leq \mathrm{Stab}_G(B)$.

Now, observe that if B is a block and $\delta, \gamma \in B$, there is $g \in G$ such that $\delta^g = \gamma$. But then $\gamma \in B^g$, thus $B = B^g$ and $g \in \mathrm{Stab}_G(B)$. Thus $\omega^{\mathrm{Stab}_G(B)} = B$.

Similarly, if $S \leq T \leq G$ and $B = \omega^T$ and $x \in \mathrm{Stab}_G(B)$ then $\omega^x = \omega^t$ for $t \in T$. Thus $xt^{-1} \in S \leq T$ and thus $x \in T$ which shows that $\mathrm{Stab}_G(\omega^T) = T$. This shows that we have a proper bijection. □

Definition II.37: A subgroup $S < G$ is called *maximal* if $S \neq G$ and there is no subgroup $S < T < G$ such that $S \neq T \neq G$.

Corollary II.38: A transitive permutation group is primitive if and only if a point stabilizer is a maximal subgroup.

Respectively: A subgroup $S \leq G$ is maximal if and only if the action on the cosets of S is primitive.

Finding Blocks

The following algorithm to find block systems is due to [Atk75]. Its heart is a method that for a given $\alpha \in \Omega$ determines the finest block system in which 1 and α are contained in the same block. By running through all possible α, we thus find all the minimal blocks.

NOTE II.39: As blocks correspond to subgroups containing the point stabilizer (and therefore form a lattice!) it is easy to build all blocks from these: If $1 \in B_1$ and $1 \in B_2$ are blocks in two different block systems, we use the same algorithm with a larger seed to find the finest block system, in which $B_1 \cup B_2$ is a subset of one block and so on.

The algorithm maintains a partition of Ω which is initialized to the seed being one cell, and all other points in a cell of their own. It then applies the following trivial observation to join cells, until a G-invariant partition is obtained:

LEMMA II.40: If B is block in a block system for G, and $\alpha, \beta \in B$ and $g \in G$ then α^g, β^g are in the same block.

To store the partition (and simplify the process of joining cells) we maintain a list r of cell representatives: Each cell is represented by one of its elements (arbitrarily chosen, e.g. as the first element of the ell which the algorithm encountered). For each point $\omega \in \Omega$ the corresponding representative $r[\omega]$ points to the representative of the cell containing ω. We call $r[\omega]$ the *label* of ω. Then joining two cells is done by simply replacing the label for elements in the second cell by labels for the first cell:

UNION(α, β)

Input: Two cells, given by their representatives α and β.
Output: The two cell are joined, representing them by the label for the first cell.

```
begin
 1: for ω ∈ Ω do
 2:   if r[ω] = β then
 3:     r[ω] := α;
 4:   fi;
 5: od;
end
```

NOTE II.41: This algorithm is of complexity $\mathcal{O}(|\Omega|)$ which is not optimal. The problem of merging classes is a standard problem in computer science ("Union-find") and (more) efficient data structures and algorithms for this task are discussed in textbooks.

With this we get the actual block system finding algorithm:

ALGORITHM II.42: This algorithm finds the finest block system, in which a block fully contains *seed* $\subset \Omega$. We call it with *seed* $= \{1, \alpha\}$ to obtain minimal blocks.

Input: A group $G = \langle \underline{g} \rangle$ acting transitively on Ω. A subset *seed* $\subset \Omega$.
Output: The finest block system in which all points of *seed* are together in one block.

```
begin
 1: r := [];
 2: q := []; {A queue of points that have changed their block}
 3: μ := seed[1]
 4: for ω ∈ Ω do
 5:   if ω ∈ seed then
 6:     r[ω] := μ;
 7:     Add ω to q;
 8:   else
 9:     r[ω] := ω;
10:   fi;
11: od;
12: l := 1;
13: while l ≤ |q| do
14:   γ := q[l]; δ := r[γ]; {point and its representative}
15:   for g ∈ g do
16:     α := r[γ^g];
17:     β := r[δ^g];
18:     if α ≠ β then {Two points are in the same block but their images are
        not}
19:       UNION(α, β); {join block given by β to block given by α}
20:       Add β to q; {As β block got deleted}
21:     fi;
22:     l := l + 1;
23:   od;
24: od;
25: return r;
end
```

Proof: Clearly the partition given by r can never be coarser than the minimal block system given by *seed*, as we only join cells that must be contained in the same block. We thus need to show only that the partition returned at the end is invariant under G, i.e. we have to show that if $\omega, \delta \in \Omega$ are in the same cell and $g \in \underline{g}$, then ω^g and δ^g are in the same cell.

This property is fulfilled if the following condition holds for all points:

(*) If β is the label for a cell, and ω is in this cell, then β^g and ω^g are in the same cell.

Clearly it is sufficient to enforce condition (*) for all points which changed the label of their cell, starting with changing the cell label for the seed. The queue q collects the points for which this condition needs to be enforced.

Suppose initially that in line 20 we add *all* points of the cell labeled by β to the queue. Then condition (*) is enforced by the `while` loop in line 13-24 and the resulting partition therefore clearly G-invariant.

However in the actual algorithm we add only β to the queue, we have to show that doing so is sufficient: Consider a point ω that is labeled by β and suppose we relabel ω to α. This can only happen if we also relabel β to α and in this case we enforce (*) for β and α.

But as ω got relabeled at the same time, and as we already enforced (*) for ω and β, this will automatically enforce (*) for ω and α. It is therefore sufficient in line 20 to only add the point labeling a block.

This argument also shows that a point ω can be added to the queue only when $r[\omega] = \omega$ gets changed to another label. As this can only happen once, there is a limit on the queue length, which proves that the algorithm terminates. □

Let us now consider what candidates for α we really need for block seeds $\{1, \alpha\}$:

Lemma II.43: Let $1 \in B \subset \Omega$ a block in a block system for G on Ω. Then B is the union of orbits of $\mathrm{Stab}_G(1)$.

Proof: Suppose there is $g \in \mathrm{Stab}_G(1)$ such that $\alpha^g = \beta$. Than for any block B such that $1, \alpha \in B$ we have that $B^g \cap B \neq \emptyset$, thus $B = B^g$. Thus also $\beta \in B$. □

This lemma shows that we do not need to test minimal blocks for all $\alpha \in \Omega$, but that it is sufficient to test those α which are representatives for the orbits of $\mathrm{Stab}_G(1)$ on Ω, and in this case we can actually seed the block with $\{1\} \cup \alpha^{\mathrm{Stab}_G(1)}$.

If we did not yet compute a stabilizer chain for G obtaining $\mathrm{Stab}_G(1)$ is hard. In this case we just approximate $\mathrm{Stab}_G(1)$ by a subgroup U generated by a few random Schreier generators and consider the orbits of U instead.

Performance II.44: Even with a better union find routine this algorithm is not of best-known complexity. A better method, interleaving the block search with a partial stabilizer chain computation, is described in [Ser03].

Once we have found a block system, the homomorphism representing the action on the blocks is obtained by an easy application of the orbit algorithm.

Basic Sylow Subgroup Computation

A first application of how blocks can be used to reduce a problem is given by the computation of Sylow subgroups: The following method works reasonably well in practice and also serves as a good example on how algorithms use reductions of intransitivity and imprimitivity. It is, however, not of polynomial time as it uses a backtrack search. A much more elaborate (polynomial time) algorithm has been proposed by Kantor [Kan85]. Because it reduces to the case of simple groups some of the routines it requires (section VII.3) just now are reaching feasibility.

The basic idea of the calculation is that if $\varphi: G \to H$ is a homomorphism to a smaller group, we first compute a p-Sylow subgroup $S^\varphi \leq H$. Its full preimage S then must contain a p-Sylow subgroup of G.

In the case of a subdirect product this is all we need:

Lemma II.45: Suppose G is a subdirect product of $A = G^\alpha$ with $B = G^\beta$. Let $Q \leq G$ be such that Q^α is a p-Sylow subgroup of A and let $\nu = \beta|_Q$ be the restriction of β to Q. Let $P \leq Q$ be such that P^ν is a p-Sylow subgroup of Q^ν. Then P is a p-Sylow subgroup of G.

Proof: Clearly Q contains a p-Sylow subgroup of G and P contains a p-Sylow subgroup of Q. Furthermore P^α and P^β are p-groups, so P is a subdirect product of p-groups. □

We will make use of this lemma in two situations: If G is intransitive (with homomorphisms corresponding to orbit actions) and if G has two different minimal block systems (with action on the blocks as homomorphisms).

If G is imprimitive and has only one one minimal block system with block action φ we can reduce to the situation that G^φ is a p group.

If we cannot reduce further, we use the fact that a p-Sylow subgroup has a non-trivial center and that (second Sylow theorem!) every element of order p lies in a Sylow subgroup: By random search we find an element $h \in G$ such that $p \mid |h|$. (It can be shown that there are many such elements.) Then $g = h^{\frac{|h|}{p}}$ is an element of order p and as such must lie in a Sylow subgroup. In fact it either (case 1) lies in the center of a Sylow subgroup, or (case 2) there is an element in the center of the same Sylow subgroup commuting with g.

We therefore compute $C := C_G(g)$. As C stabilizes the partition of Ω into orbits of $\langle g \rangle$ it cannot be primitive. We therefore can compute recursively a p-Sylow subgroup S of C.

As $g \in Z(C)$, it must lie in every Sylow subgroup of C, in particular in S.

In case 1 we have that C must contain a Sylow subgroup of G and so S is the desired result.

Otherwise (case 2) there exists a P-Sylow subgroup P of G such that $g \in P$ and there is $z \in Z(P)$ of order p. Thus z commutes with g which implies that $z \in C$.

Now $P \cap C$ is a p-group with $S \leq P \cap C \leq C$. As S is a Sylow subgroup of C we must have equality $S = P \cap C$, implying that $z \in S \leq P$. As z commutes with all elements of P it commutes with S, i.e. $z \in Z(S)$ and $P \leq C_G(z)$.

We thus search for an element of order p in $Z(S)$ for which $C_G(z)$ contains a p-Sylow subgroup of G and find it recursively in $C_G(z)$.

Algorithm II.46: Sylow subgroup computation

Input: A group G on Ω and a prime p
Output: A p-Sylow subgroup $S \leq G$.
begin

if G is a p-group **then**
  return G
elif G is intransitive on Ω **then**
  recurse on orbit actions, using lemma II.45
elif $p \nmid |\Omega|$ **then**
  recurse on $\mathrm{Stab}_G(1)$
elif G has two minimal block systems **then**
  recurse on block actions action, using lemma II.45
elif G has unique minimal block system **then**
  ensure (recursively) the image of block action of G is a p-group
fi;
let $h \in G$ such that $p \mid |h|$ and set $g = h^{\frac{|h|}{p}}$.
if $p^2 \nmid |G|$ **then**
  return $\langle g \rangle$
fi;
Let $C = C_G(g)$;
Recursively, compute a p-Sylow subgroup S of C.
if $p \nmid \left[G{:}C\right]$ **then**
  return S;
fi;
Let $Z = Z(S)$ {iterative centralizer computation}
for $z \in Z$, $|z| = p$ **do**
  $C := C_G(z)$;
  if $p \nmid \left[G{:}C\right]$ **then**
    recurse on C
  fi;
od;
end

Wreath Products and the Embedding theorem

In the same way that every intransitive group is a subgroup of a direct product, we want to get an "universal" group containing every imprimitive group.

Definition II.47: If G is a group and m a positive integer we denote by

$$G^{\times n} := \underbrace{G \times \cdots \times G}_{n \text{ times}}$$

the direct product of n copies of G. We call thi group the *direct power* of G with exponent m

Definition II.48: Let G be a group and H a permutation group, acting on $\Delta = \{1, \ldots, n\}$. The *wreath product* of G with H is

$$G \wr_n H = (G^{\times n}) \rtimes H$$

with H acting on $G^{\times n}$ by permuting the components of this direct product. The subgroup $G^{\times n} \lhd G \wr_n H$ is called the *basis* of the wreath product.

If the permutation action of H is clear from the context, we will write only $G \wr H$.

Suppose that G is also a permutation group, acting on Ω. Then $G \wr H$ can be represented as a permutation group acting on n disjoint copies of Ω: Each copy of G in the basis acts on "its" copy of Ω, H is permuting these copies. The action is clearly faithful. We call this action the *imprimitive action* of $G \wr H$, as the copies of Ω form a nontrivial block system.

Theorem II.49 (Krasner, Kaloujnine, embedding theorem): Let G be a transitive, imprimitive permutation group. Let $\mathcal{B}$ be a nontrivial block system for G with $1 \in B \in \mathcal{B}$ and let $T = \mathrm{Stab}_G(B)$. Let $\psi: G \to S_{\mathcal{B}}$ be the action of G on the blocks, and let $\varphi: T \to S_B$ be the action of a block stabilizer on its block.

We pick coset representatives r_j for T in G and define $\widetilde{g_j} \in T$ by $r_j g = \widetilde{g_j} r_{j^g}$.

Then there is a monomorphism $\mu: G \to T^{\varphi} \wr G^{\psi}$, given by

$$g \mapsto (\widetilde{g_{1^{g^{-1}}}}^{\varphi}, \ldots, \widetilde{g_{n^{g^{-1}}}}^{\varphi}; g^{\psi})$$

Note II.50: In the context of representation theory μ is simply the induced representation $\varphi \uparrow^G$. The theorem then is simply the explicit construction of the induced representation.

Proof: Explicit check. □

Note II.51: There unfortunately is no analogue to the situation of subdirect products, that would parameterize all transitive, imprimitive subgroups of a wreath product. An algorithm to construct such subgroups is given in [Hul05]

II.5 Primitive Groups

Primitive groups are interesting in several ways: They are the images of the permutation action of a group on cosets of maximal subgroups. By theorem II.49 we also know that every transitive group embeds in an (iterated) wreath product of primitive groups.

The marvelous fact now is that primitivity is a strong enough condition to give a rather detailed description of such groups. Indeed this description is strong enough, that it is possible to enumerate primitive groups for rather large degrees – currently this has been done up to degree 2000 [DM88, The97, RDU03].

Some Properties

Lemma II.52: Let G be a transitive group on Ω and $N \lhd G$. Then the orbits of N form a block system of G

Proof: Let Δ be an orbit of N and $g \in G$. We need to show that Δ^g is a subset of an orbit. (If this holds and Δ^g was not an orbit, we can apply g^{-1} to the enclosing orbit and obtain that Δ was a proper subset of an orbit, contradiction.) Thus let $\delta^g, \gamma^g \in \Delta^g$ for $\delta, \gamma \in \Delta$. Then there is $n \in N$ such that $\delta^n = \gamma$ and thus $(\delta^g)^{g^{-1}ng} = \gamma^g$. □

Corollary II.53: If G is primitive on Ω then N must act transitively.

The heart of the analysis will be the consideration of particular normal subgroups:

Definition II.54: A normal subgroup $N \triangleleft G$ is called minimally normal, if $\langle 1\rangle \neq N$ and there is no normal subgroup $M \triangleleft G$ such that $\langle 1\rangle \neq M \neq N$ and $\langle 1\rangle < M < N$.

Lemma II.55: Let G be a group and $N \triangleleft G$ a minimally normal subgroup. Then $N \cong T^{\times k}$ with T simple.

Proof: Let $M \triangleleft N$ be the first proper subgroup in a composition series of N. Then $N/M \cong T$ is simple. Now consider the orbit $\mathcal{M} = M^G$ of M under G. Let $D := \bigtimes_{S\in\mathcal{M}} N/S$ and $\varphi\colon N \to D$, $g \mapsto (S_1 g, S_2 g, \ldots)$. Its kernel is $K := \bigcap_{S\in\mathcal{M}} S \triangleleft G$, by minimality of N we get that $K := \langle 1\rangle$. Thus φ is injective and N a subdirect product of the groups N/S ($S \in \mathcal{M}$). But $N/S \cong T$ is simple, thus the subdirect product degenerates (by problem **??**) to a direct product. □

Definition II.56: The *socle* of a group G is the subgroup generated by all minimal normal subgroups:

$$\mathrm{Soc}(G) = \langle N \triangleleft G \mid N \text{ is minimally normal}\rangle$$

Lemma II.57: $\mathrm{Soc}(G)$ is the direct product of minimal normal subgroups.

Proof: Let $M \leq \mathrm{Soc}(G)$ be the largest normal subgroup of G within $\mathrm{Soc}(G)$ which is a direct product of minimal normal subgroups. If $M \neq \mathrm{Soc}(G)$ there exists $N \triangleleft G$, minimally normal, such that $N \not\leq M$. But then $M \cap N \triangleleft G$. As N is minimally normal this implies that $M \cap N = \langle 1\rangle$. Thus $\langle M, N\rangle = M \times N \leq \mathrm{Soc}(G)$, contradicting the maximality of M. □

Next we want to show that for a primitive group $\mathrm{Soc}(G)$ is either minimally normal, or the product of two isomorphic minimally normal subgroups:

Definition II.58: A permutation group G is *semiregular* on Ω if $\mathrm{Stab}_G(\omega) = \langle 1\rangle$ for every $\omega \in \Omega$.

Thus G is regular on Ω if and only if G is transitive and semiregular.

Lemma II.59: Let $G \leq S_\Omega$ be transitive. Then $C := C_{S_\Omega}(G)$ is semiregular.

Proof: Suppose that $c \in \mathrm{Stab}_C(\omega)$ and let $\delta \in \Omega$. Then there is $g \in G$ such that $\delta = \omega^g = \omega^{cg} = \omega^{gc} = \delta^c$, thus $c \in \mathrm{Stab}_C(\delta)$ for every δ. Thus $c = 1$. □

Lemma II.60: Let G be a primitive group on Ω. Then one of the following situations holds:

a) $\mathrm{Soc}(G)$ is minimally normal

b) $\mathrm{Soc}(G) = N \times M$ with $N, M \triangleleft G$ minimally normal and $N \cong M$ is not abelian.

Proof: Suppose that $\mathrm{Soc}(G)$ is not minimally normal. By lemma II.57 we have that $\mathrm{Soc}(G) = N \times M$ with $N \triangleleft G$ minimally normal and $\langle 1\rangle \neq M \triangleleft G$. Then $M \leq C_G(N)$ and $N \leq C_G(M)$.

As G is primitive, N is transitive on Ω. Thus by lemma II.59 we have that M must be semiregular. On the other hand $M \triangleleft G$ implies that M is also transitive on Ω, thus N is semiregular. In summary thus both N and M must be regular, and thus $|N| = |\Omega| = |M|$.

For $n \in N$ there exists a unique element $m_n \in M$ such that $(1^n)^{m_n} = 1$. Let $\varphi\colon N \to M$ given by $n \mapsto m_n$. Then φ is clearly a bijection. Furthermore (using that $M, N \leq S_\Omega$) for $k, n \in N$:

$$1^{k\cdot n\cdot m_k\cdot m_n} = 1^{k\cdot m_k\cdot n\cdot m_n} = \left((1^k)^{m_k}\right)^{n\cdot m_n} = 1^{n\cdot m_n} = 1$$

and therefore $m_k m_n = m_{kn}$. Thus φ is an isomorphism.

If N was abelian, then $N \times M$ is abelian and transitive, thus $|N \times M| = |\Omega|$, contradiction. □

We thus have that $\mathrm{Soc}(G) \cong T^{\times m}$ with T simple. We say that $\mathrm{Soc}(G)$ is *homogeneous of type* T.

Theorem II.61: Let G be primitive on Ω and $\mathrm{Soc}(G)$ abelian. Then $|\Omega| = p^m$ for some prime p and $G = \mathrm{Soc}(G) \rtimes \mathrm{Stab}_G(1)$ with $\mathrm{Stab}_G(1)$ acting (by conjugation) irreducibly and faithfully on $\mathrm{Soc}(G) \cong \mathbb{F}_p^m$.

Proof: If $\mathrm{Soc}(G)$ is abelian, it is minimally normal and thus $\mathrm{Soc}(G) \cong \mathbb{F}_p^m$. It must act regularly (the only faithful transitive action of an abelian group is the regular action), thus $|\Omega| = p^m$.

Now consider $S := \mathrm{Stab}_G(1)$. Clearly $\mathrm{Soc}(G) \not\leq S$. As $S < G$ is a maximal subgroup we thus have that $G = \mathrm{Soc}(G)S$. As $\mathrm{Soc}(G)$ is abelian, $S \cap \mathrm{Soc}(G) \triangleleft \mathrm{Soc}(G)$. Also $S \cap \mathrm{Soc}(G) \triangleleft S$. Thus $S \cap \mathrm{Soc}(G) \triangleleft G$ and therefore $S \cap \mathrm{Soc}(G) = \langle 1\rangle$. This shows that G is a semidirect product.

If S was not acting irreducibly on $\mathrm{Soc}(G)$ let $T \leq \mathrm{Soc}(G)$ be a proper submodule. Then T is normalized by S and $T \triangleleft \mathrm{Soc}(G)$, thus $T \triangleleft G$ contradicting the fact that $\mathrm{Soc}(G)$ is minimally normal.

The kernel of the action of S on $\mathrm{Soc}(G)$ is contained in $C_G(\mathrm{Soc}(G)) = \mathrm{Soc}(G)$, thus the action is faithful. □

It is easily seen that vice versa any such semidirect product acts primitively on $\mathbb{F}_p^m$.

COROLLARY II.62: Let G be a solvable group and $M < G$ a maximal subgroup. Then $\left[G{:}M\right] = p^m$ for a prime p.

Proof: The image of the action of G on the cosets of M is a primitive group with an abelian minimal normal subgroup. □

LEMMA II.63: Let G be a group such that $Z(\mathrm{Soc}(G)) = \langle 1\rangle$. Then $G \le \mathrm{Aut}(\mathrm{Soc}(G))$.

Proof: Consider the action of G by conjugation on $\mathrm{Soc}(G)$. The kernel of this action is $C_G(\mathrm{Soc}(G)) \lhd G$. A minimal normal subgroup contained in $C_G(\mathrm{Soc}(G))$ would be in $Z(\mathrm{Soc}(G))$, which is trivial. Thus this action is faithful. □

LEMMA II.64: If $N = T^{\times m}$ with T non-abelian simple, then $\mathrm{Aut}(N) = \mathrm{Aut}(T) \wr S_m$

Proof: Let T_i be the i-th direct factor. Let $\varphi \in \mathrm{Aut}(N)$. Let $R := T_1^{\varphi}$. Then $R \lhd N$. Consider some nontrivial element of R as element of a direct product $r = (t_1, \ldots, t_m)$ with $t_i \in T_i$. Suppose that $t_j \neq 1$ for some j. As $Z(T_j) = \langle 1\rangle$ there exists $y \in T_j$ such that that $t_j^y \neq t_j$. Set $s := r^y/r \neq 1$. Then $s \in R$ and $s \in T_j$. As T_j is simple and $R \lhd N$ we thus get that $T_j \le R$, thus $R = T_j$.

This shows that every automorphism of N permutes the T_i. An automorphism that fixes all T_i then must act on every T_i as an element of $\mathrm{Aut}(T_i) = \mathrm{Aut}(T)$. □

COROLLARY II.65: Let G be primitive with $\mathrm{Soc}(G)$ non-abelian of type T. Then we can embed $G \le Aut(T) \wr S_m$.

Types

We now want to describe some important classes of primitive groups:

The first class is a different action of wreath products: Let G be a permutation group on Ω and H a permutation group on Δ. So far we have had the wreath product $W := G \wr H$ act (imprimitively) on $\Omega \times \Delta$. We now define a different action of W on Ω^Δ. This is a much larger set. Surprisingly, the action will turn out to be primitive in many cases.

The action is easiest described if (assume that $|\Delta| = d$) we consider Ω^Δ as a d-dimensional cube each side of which is labeled by Ω. We then let $G^{\times d}$ act independently in each dimension and H permute the dimensions. That is, we define

$$(\omega_1, \ldots, \omega_d)^{(g_1,\cdots g_d;h)} := (\omega_{1'}^{g_{1'}}, \ldots, \omega_{d'}^{g_{d'}}) \qquad \text{with} \qquad i' = i^{h^{-1}}.$$

an easy, but tedious calculation shows that this indeed is a group action, which is called the *product action*. We note that in this action the base group G^d acts transitively.

THEOREM II.66: Suppose that Ω, Δ are finite. Then $G \wr H$ in the product action is primitive if and only if G is primitive, but nonregular on Ω and H transitive on Δ.

NOTE II.67: We do not require G to be "minimal" in this theorem. Essentially, using the fact that $(A \wr B) \wr C = A \wr (B \wr C)$, we could enforce this by increasing H.

For the second class of examples, consider a socle of the form $T^{\times m}$ with T simple. Let D be a diagonal subgroup $\{(t, t, \ldots, t) \mid t \in T\}$. We consider the action of the socle on the cosets of D (of degree $n = |T|^{m-1}$).

As we want to extend this permutation action to a primitive group, we next consider the normalizer $N = N_{S_n}(T^{\times m})$. Clearly all elements of N induce automorphisms of $T^{\times m}$, however the following lemma show that not all automorphisms can be realized within S_n:

LEMMA II.68: Let $G \le S_n$ be a transitive group and $\varphi \in \mathrm{Aut}(G)$. Then φ is induced by $N_{S_n}(G)$ (i.e. here exists $h \in S_n$ such that $g^\varphi = g^h$ for every $g \in G$, obviously $h \in N_{S_n}(G)$ in this case) if and only if $\mathrm{Stab}_G(1)^\varphi = \mathrm{Stab}_G(j)$ for some $1 \le j \le n$.

Proof: Problem **??** □

Using this lemma, one sees that not all elements of $\mathrm{Aut}(T) \wr S_m$ are induced by permutations, in fact outer automorphisms need to act simultaneously on all components in the same way. Thus $N = T \wr S_m$. $\mathrm{Out}(T) = T^{\times m} \rtimes (S_m \times \mathrm{Out}(T))$ with the outer automorphisms acting simultaneously on all components. Such a group $T^{\times m} \le G \le N$ is said to be of *diagonal type*.

THEOREM II.69: A group of diagonal type is primitive if $m = 2$ or the action of G on the m copies of T is primitive.

The O'Nan-Scott Theorem

We now can state a theorem that classifies the structure of all primitive groups. The theorem was stated first (with a small error) by L. SCOTT in 1979. (In principle it would have been possible to prove this theorem 50 years earlier, but the reduction to the simple case only made sense with the classification of the finite simple groups.) He notes that a similar theorem was obtained by M. O'NAN, thus the name.

THEOREM II.70 (O'NAN-SCOTT theorem): Let G be a group which acts primitively and faithfully on Ω with $|\Omega| = n$. Let $H = \mathrm{Soc}(G)$ and $\omega \in \Omega$. Then $H \cong T^{\times m}$ is homogeneous of type T for T simple and exactly one of the following cases holds.

1. "Affine". T is abelian of order p, $n = p^m$ and $\mathrm{Stab}_G(\omega)$ is a complement to H which acts irreducibly on H.
2. "Almost simple". $m = 1$ and $H \lhd G \le \mathrm{Aut}(H)$.
3. "Diagonal type". $m \ge 2$ and $n = |T|^{m-1}$. Further, G is a subgroup of $V = (T \wr S_m)$. $\mathrm{Out}(T) \le \mathrm{Aut}(T) \wr S_m$ in diagonal action and either

a) $m = 2$ and G acts intransitively on $\{T_1, T_2\}$ or

b) $m \geq 2$ and G acts primitively on $\{T_1, \ldots, T_m\}$.

In case a) T_1 and T_2 both act regularly. Moreover, the point stabilizer V_ω of V is of the form $\mathrm{Diag}(\mathrm{Aut}(T)^{\times m}).S_m \cong \mathrm{Aut}(T) \times S_m$ and thus $H_\omega = \mathrm{Diag}(T^{\times m})$.

4. "Product type". $m = rs$ with $s > 1$. We have that $G \leq W = A \wr B$ and the wreath product acts in product action with A acting primitively, but not regularly, on d points and B acting transitively on s points. Thus $n = d^s$. The group A is primitive of either

   a) type 3a with socle $T^{\times 2}$ (i.e. $r = 2$, $s < m$),

   b) type 3b with socle $T^{\times r}$ (i.e. $r > 1$, $s < m$) or

   c) type 2 (i.e. $r = 1$, $s = m$).

   We have that $W_\omega \cap A^s \cong A_1^{\times s}$ and $\mathrm{Soc}(G) = \mathrm{Soc}(W)$. Furthermore $W = A^{\times s}G$.

5. "Twisted wreath type". H acts regularly and $n = |T|^m$. G_ω is isomorphic to a transitive subgroup of S_m. The normalizer $N_{G_\omega}(T_1)$ has a composition factor isomorphic to T. Thus, in particular, $m \geq k+1$ where k is the smallest degree of a permutation group which has T as a composition factor.

Note II.71: We do not discuss the twisted wreath case in detail, but note that the minimum degree for this is 60^6.

The proof of this theorem is not extremely hard (see for example [DM96]), but would take us about 3 lectures.

Note II.72: There are various versions of this theorem in the literature which in particular differ by the labeling of the cases and sometimes split cases slightly differently. Our version follows [DM96] and in particular [EH01].

The following table gives translations of the labellings used.

Type	1	2	3a	3b	4a	4b	4c	5
[HEO05, Sec.10.1.3]	(i)	(ii)a, d=1	(ii)b	(iii)	(ii)b	(iii)	(ii)b	(ii)c
[DM96, Sec.4.8]	i	iii	iv	iv	v	v	v	ii
[LPS88]	I	II	IIIa	IIIa	IIIb	IIIb	IIIb	IIIc
[Neu86]	I	V	II	III	II	III	IV	IV

Note that for case 3a/b we change the case distinction of [DM96, Theorem 4.5A] from degree 2/> 2 to intransitive/primitive.

Maximal subgroups of the Symmetric Group

Most classes in the O'Nan-Scott theorem contain obvious maximal elements. Every primitive group thus is contained in such a maximal element.

As one can show that these subgroups are not just maximal in their classes, but also maximal in the symmetric group, we get a classification of maximal subgroups of the symmetric group:

Theorem II.73: Let $M \leq S_n$ be a maximal subgroup of the symmetric group. Then M is permutation isomorphic to one of the following groups:

- A_n
- $S_a \times S_b$ with $a + b = n$.
- $S_l \wr S_m$ in imprimitive action for $lm = n$.
- $AGL_m(p)$ with $n = p^m$
- $S_a \wr S_b$ in product action for $n = a^b$
- $T^{\times a}.(S_a \times \mathrm{Out}(T))$ with T simple and $n = |T|^{a-1}$
- $T \leq G \leq \mathrm{Aut}(T)$ for a simple group T

Note II.74: For some degrees there are inclusions among elements in these classes, but these occur very rarely. A full classification is given in [LPS87].

II.6 Computing a Composition Series

The basic idea of finding a composition series in a permutation group is very easy:

> Given a permutation group G, either prove that G is simple; or find – from a suitable action – a homomorphism (which we can evaluate by performing the action) $\varphi: G \to H$ such that H is a permutation group of degree smaller than that of G or $N := \mathrm{Kern}\,\varphi > \langle 1 \rangle$.

If we can solve this problem, we can recursively attack N and $G^\varphi \cong G/N$ until we end up with simple composition factors. Pulling the factors of G/N back through φ yields a composition series.

If G is an intransitive permutation group we can take for φ the action of G on one orbit. If G is imprimitive we can take for φ the action on a nontrivial block system. (Either these actions already yield a nontrivial kernel, or they yield a group of smaller degree for which we try again.)

Thus what remains to deal with is the case of G primitive and for such groups the O'Nan-Scott theorem provides structure information. Our main aim will be to find $\mathrm{Soc}(G)$. Then the action of G on $\mathrm{Soc}(G)$ or on the components of $\mathrm{Soc}(G)$ yields homomorphisms with nontrivial kernel.

The affine case

The first case we want to treat is the case of G being primitive affine. In this case the socle is an elementary abelian regular normal subgroup, often abbreviated as EARNS. Given G, we therefore want to find an EARNS in G if it exists. The algorithm for this is due to [Neu86].

Clearly we can assume that G is a group of prime-power degree $n = p^m = |\Omega|$.

We first consider two special cases:

If $\mathrm{Stab}_G(\alpha) = 1$ for $\alpha \in \Omega$, then G is regular, and thus of prime order. It is its own EARNS.

Definition II.75: A transitive permutation group G on Ω is called a *Frobenius group* if for every $\alpha, \beta \in \Omega$ ($\alpha \neq \beta$) the two-point stabilizer $\mathrm{Stab}_G(\alpha, \beta) = \langle 1 \rangle$.

A classical (1902) result of Frobenius shows that in our situation G must have an EARNS (for a proof see [Pas68]). As $|G| \leq n(n-1)$ this is easily found. (See [Neu86] for details.)

No suppose we are in neither of these cases. Let $\alpha, \beta \in \Omega$. We consider the two-point stabilizer $G_{\alpha\beta} := \mathrm{Stab}_G(\alpha, \beta) \neq \langle 1 \rangle$. By choosing β from an orbit of $\mathrm{Stab}_G(\alpha)$ of shortest length, we can assume that $G_{\alpha\beta}$ is as large as possible.

Let $\Delta = \{\omega \in \Omega \mid \omega^g = \omega \forall g \in G_{\alpha\beta}\}$. If G has an EARNS N, then for $\gamma \in \Delta$ there is a unique $n \in N$ such that $\alpha^n = \gamma$. Then for $h \in G_{\alpha\beta}$ we have that

$$\underbrace{h^{-1}n^{-1}h}_{\in N \lhd G} n = \underbrace{h^{-1}}_{\in G_{\alpha\beta} \leq \mathrm{Stab}_G(\gamma)} \cdot \underbrace{n^{-1}hn}_{\in \mathrm{Stab}_G(\gamma)} \in N \cap \mathrm{Stab}_G(\gamma) = \langle 1 \rangle$$

and thus $n \in C := C_G(G_{\alpha\beta})$.

In particular C acts transitively on Δ, $N \cap C$ acts regularly on Δ, thus $|\Delta|$ must be a p-power (if either of this is not the case, G has no EARNS).

Now consider the homomorphism $\varphi: C \to S_\Delta$, then the image C^φ is transitive on Δ. The kernel of φ consists of elements that stabilize all points in Δ (in particular α and β) and centralize $G_{\alpha\beta}$, thus $\mathrm{Kern}\,\varphi \leq Z(G_{\alpha\beta})$.

For $\gamma \in \Delta \setminus \{\alpha\}$ we have that $G_{\alpha\beta} \leq \mathrm{Stab}_G(\alpha, \gamma)$, but as β was chosen to yield a maximal stabilizer, we get equality. Thus $\mathrm{Stab}_C(\alpha\gamma)^\varphi = \langle 1 \rangle$ and C^φ is a Frobenius group.

Let $K \leq C$ such that $K^\varphi = (N \cap C)^\varphi$ is the EARNS of C^φ. Thus $N \cap K \neq \langle 1 \rangle$ and we just need to get hold of some element in $N \cap K$ to find N.

We claim that K is abelian: This is because K is generated by $\mathrm{Kern}\,\varphi \leq Z(G_{\alpha\beta})$ and by elements of $N \cap C$ which commute with each other (N is abelian) and with $\mathrm{Kern}\,\varphi \leq G_{\alpha\beta}$ (as they are in C).

Next compute $P = \{x \in \mathrm{Kern}\,\varphi \mid x^p = 1\}$. (As $\mathrm{Kern}\,\varphi$ is abelian, this is easy.)

We now find $x \in K \setminus \mathrm{Kern}\,\varphi$ such that $|x| = p$.

Then $1 \neq x^\varphi = g^\varphi$ for some $g \in N \cap K \leq$ and $x^{-1}g \in \mathrm{Kern}\,\varphi$. As K is abelian $|x^{-1}g| = p$, thus $x^{-1}g = h \in P$ and $g \in N \cap Px$.

We thus run through the elements $h \in P$, and test whether $\langle xh \rangle_G$ is abelian and regular – if so it is the EARNS.

Note II.76: A variant of this method can be used to find in a regular normal subgroup of G also for type 5 groups.

Note II.77: Variants of the method can be used to find the largest normal p-subgroup $O_p(G) \lhd G$ and the *radical* of $O_\infty(G) \lhd G$, which is the largest solvable subgroup of G. These methods also construct a homomorphism from G to a group of not larger degree such that the kernel is $O_p(G)$, respectively $O_\infty(G)$

Finding the socle and socle components

The following methods follow [Neu87]. They differ from what is used in practice but give an idea of the methods and are easier to describe:

Theorem II.78 (Schreier's conjecture): Let G be a simple non-abelian group. Then $\mathrm{Aut}(G)/G$ is solvable of derived length ≤ 3.

Proof: Inspection, following the classification of finite simple groups. □

Lemma II.79: Let G be a primitive group with no nontrivial abelian normal subgroup. Let $S := \mathrm{Soc}(G) = T_1 \times \cdots \times T_m$ with $T_i \cong T$ simple non-abelian. Let $U \leq G$ be a 2-Sylow subgroup and $N = \langle Z(U) \rangle_G$. Then $S = N'''$.

Proof: By Feit-Thompson $2 \mid |T_i|$. As T_i is subnormal in G, we know that $U \cap T_i \neq \langle 1 \rangle$. Thus every element of $Z(U)$ must centralize some elements in T_i. Considering G embedded in $\mathrm{Aut}(T) \wr S_m$ we thus see that elements of $Z(U)$ may not move component i. Thus $Z(U) \leq \mathrm{Aut}(T)^m \cap G \lhd G$. Thus $\langle Z(U) \rangle_G \leq \mathrm{Aut}(T)^m \cap G$. But $(\mathrm{Aut}(T)^m)''' = T^{\times m}$ by theorem II.78.

On the other hand, $Z(U \cap T_i) \neq \langle 1 \rangle$ and (as those elements commute with all other T_j and with $U \cap T_i$, we have that $Z(U \cap T_i) \leq Z(U)$. Thus $Z(U) \cap T_i \neq \langle 1 \rangle$, which shows that $T^{\times m} \leq \langle Z(U) \rangle_G$. □

Using this lemma we easily obtain $\mathrm{Soc}(G)$.

Note that the only case in which $\mathrm{Soc}(G)$ can be primitive itself is in diagonal action for $m = 2$. In this case it is not hard to find an element in T_i (just take a maximal nontrivial power of a random element try the normal closure).

Otherwise we can use further reduction to imprimitivity/intransitivity to find the socle components.

Chief Series

Computing a chief series is not much harder. The only difference is that we always have to ensure normality in the whole group. We can do this by simply intersecting conjugates.

LEMMA II.80: Suppose that $N \lhd G$ and $M \lhd N$ with N/M simple. Then $L := \bigcap_{g \in G} M^g \lhd G$. If N/M is non-abelian then N/L is a minimal normal subgroup of G/L.

Proof: Homework □

If N/M (and thus N/L) is (elementary) abelian, we use Meataxe-type methods to reduce to chief factors.

In practice one would first compute the radical $R := O_\infty(G)$. Since this was obtained from iterated computations of p-cores $O_p(G)$ we have in fact already some splitup of R into chief factors and finish using the Meataxe.

The radical factor G/R then is treated using reduction to orbits, block systems etc.

II.7 Other groups with a natural action

There is a variety of other groups, which have naturally a faithful permutation action and thus could be treated like permutation groups. For example:

- Matrix groups $G \leq GL_n(p)$. Here the action is on vectors in $\mathbb{F}_p^n$.
- Groups of group automorphisms $G \leq \mathrm{Aut}(H)$. The action is on elements of H.

NOTE II.81: We could (using the orbit algorithm) simply compute an isomorphic permutation group. However the permutation degree then tends to be large and for memory reasons it is often convenient to keep the original objects.

There is however one fundamental problem, for example for stabilizer chains: In general these groups have few short orbits. Thus, just picking random base points, will likely lead to very long orbits, often even to regular orbits (i.e. the first stabilizer is already trivial).

One can try to invest some work in finding short orbits (for matrix groups, for example base points that are eigenvector of random matrices or subgroups generated by random matrices have been proposed [MO95], as they guarantee the orbit to be not regular). In general, however this will not be sufficient.

Instead we consider *additional*, different, actions of G, which are related to the original action, but are not necessarily faithful. If H is the image of such an action, we would consider the permutation group $G \wr H$ with the whole factor group H glued together (so abstractly, the group is isomorphic G), acting intransitively. We then pick base points initially from the points moved by H, thus obtaining smaller orbit lengths. Once we need to pick base points from the original domain, we have a smaller group which automatically yields shorter orbits.

Since the second action can be obtained from the first, we do not really need to write down this pseudo-subdirect product, but simply consider different actions.

In terms of defining a stabilizer chain, each layer of the chain simply carries a description of the appropriate action. Furthermore, we might switch the action multiple times in one stabilizer chain.

If G is a matrix group (over a field F of size > 2) an obvious related action is to act projectively, i.e. on 1-dimensional subspaces instead on vectors. This will typically reduce the initial orbit length by a factor of $|F - 1|$.

If G is a group of automorphisms of another group H, one can determine a characteristic (i.e. fixed under all automorphisms) subgroup $N \lhd H$ and initially consider the induced actions on N and on H/N.

Incidentally, it can be useful to do something similar for permutation groups: If the group is imprimitive, consider first the action on the blocks to get much shorter orbits.

II.8 How to do it in GAP

Stabilizer Chains

Backtrack

Blocks and primitivity

Subdirect products and wreath products

Composition series and related functions

Matrix groups and automorphism groups

Chapter

III

Finitely presented groups

Finitely presented groups are probably the most natural way to describe groups. Unfortunately they also are the computational least tractable and only afford a restricted set of methods.

In this chapter and the following we will often have to talk about generating systems, and about words (product expressions) in a particular generating system. If $\underline{\boldsymbol{g}}$ and $\underline{\boldsymbol{h}}$ are two sequences of elements of the same cardinality, and $w(\underline{\boldsymbol{g}})$ is a product of elements of the one generating system, then we will write $w(\underline{\boldsymbol{h}})$ to mean the same product expression, but with every g_i replaced by h_i.

III.1 What are finitely presented groups

Free Groups

Definition III.1: A group $F = \left\langle \underline{\boldsymbol{f}} \right\rangle$ is *free* on the generating set $\underline{\boldsymbol{f}}$ if every map $\underline{\boldsymbol{f}} \to H$ into a group H can be extended to a homomorphism $F \to H$.

Note III.2: This is a property the basis of a vector space has.

It is not hard to show that the isomorphism type of a free group is determined by the cardinality of the generating system, we therefore will usually talk about a *free group of rank m*.

We now want to show that free groups exist. For this we consider a set of m letters: $x_1, x_2, \ldots, x_m$. (Or, if one prefers, $a, b, c, \ldots$.) We add m extra symbols $x_1^{-1}, \ldots, x_m^{-1}$, we call the resulting set of symbols our *alphabet A*.

For this alphabet A we consider the set A^* of *words* (i.e. finite sequences of letters, including the empty sequence) in A. Next we introduce an equivalence relation $\frown$ on A^*: Two words in A are said to be directly equivalent, if one can be obtained from the other by inserting or deleting a sequence $x \cdot x^{-1}$ or $x^{-1}x$. We define $\frown$

as the equivalence relation (smallest classes) on A^* induced by direct equivalence. We now consider $F = A^*/\frown$. On this set we define the product of two classes as the class containing a concatenation of representatives. One then can show:

Theorem III.3: a) This product is well-defined.
b) F is a group.
c) F is free on $x_1, \ldots, x_n$.

Proof: Tedious calculations: The hardest part is associativity as we have to consider multiple cases of cancellation. □

Note III.4: By performing all cancellations, there is a shortest representative for every element of F, we will simply use these representatives to denote elements of F.

Presentations

Now suppose that F is a free group of rank m. Then every group generated by m elements is isomorphic to a quotient F/N. We want to describe groups in such a way by giving a normal subgroup generating set for N.

Definition III.5: A *finitely presented group* G is a quotient $F/\langle R \rangle_F \cong G$ for a finite subset $R \subset F$. If $\underline{\boldsymbol{g}}$ is a free generating set for F we write $G \cong \left\langle \underline{\boldsymbol{g}} \mid R \right\rangle$ to describe this group and call this a *presentation* of G.
We call the elements of R a set of defining *relators* for G.

Instead of relators one sometimes considers *relations*, written in the form $l = r$. We will freely talk about relations with the interpretation that the corresponding relator l/r is meant.

Note III.6: In general there will be many different presentations describing the same group.

Note III.7: Besides being a convenient way for describing groups, finitely presented groups arise for example naturally in Topology, when describing the fundamental group of a topological space.

Lemma III.8: Every finite group is finitely presented

Proof: Suppose $|G| < \infty$. Choose a map $\varphi: F_{|G|} \to G$ that maps generators to the elements of G. It extends to a surjective homomorphism. Kern φ has finite index in $F_{|G|}$ and thus a finite number of Schreier generators. □

In this chapter we want to study algorithms for (finite or infinite) finitely presented groups.

Note III.9: We will typically represent elements of a finitely presented group by their representatives in the free group, but we should be aware that these represen-

tatives are not unique. Also there is in general no easy "normal form" as there is for small examples. (See chapter IV for more information about this.)

III.2 Tietze Transformations

There are some simple modifications of a presentation that do not change the group. They are called *Tietze Transformations*:

LEMMA III.10: Suppose we have a presentation $G = \left\langle \underline{\boldsymbol{g}} \mid R \right\rangle$. Then the following transformations (called "Tietze Transformations") do not change G:

1. Add an extra relator that is a word in R.
2. Delete a relator that can be expressed as a word in the other relators.
3. For a word w in $\underline{\boldsymbol{g}}$, add a new generator x to $\underline{\boldsymbol{g}}$ and a new relator $x^{-1}w$ to R
4. If a generator $x \in \underline{\boldsymbol{g}}$ occurs only once and only in one relator, delete x and delete this relator.

Proof: Transformations 1 and 2 obviously do not change $\langle R \rangle_F$. For Transformations 3 and 4 there is an obvious map between the old and new groups, which preserves all relators and thus is an isomorphism. □

Tietze transformations were defined in the context of the following

LEMMA III.11: Suppose that the presentations $P_1 = \left\langle \underline{\boldsymbol{g}} \mid R \right\rangle$ and $P_2 = \langle \underline{\boldsymbol{h}} \mid S \rangle$ yield isomorphic groups. Then there is a sequence of Tietze transformations from P_1 to P_2.

Proof: (Idea) If there is an isomorphism between P_1 and P_2 go first from P_1 to $Q = \left\langle \underline{\boldsymbol{g}} \cup \underline{\boldsymbol{h}} \mid R \cup S \cup T \right\rangle$ by adding relators T that express $\underline{\boldsymbol{h}}$ in terms of $\underline{\boldsymbol{g}}$ and deduce the relators in S, then delete the redundant $\underline{\boldsymbol{g}}$ by expressing them as words in $\underline{\boldsymbol{h}}$ to go from Q to P_2. □

This lemma itself is of little use, as the path of transformations between presentations is not known, it is not even known to be bounded in length.

They can however be used heuristically to try to simplify a presentation:

Only apply transformations which make a presentation immediately more simple; either by removing or shortening relators or by removing generators without increasing the overall length of the relators too much.

NOTE III.12: By combining transformations, we get the following transformations which are more useful:

1. Replace a relator r by $x^{-1}rx$ for $x \in \underline{\boldsymbol{g}}$. In particular, if $r = xa$ starts with x, this yields the cyclically permuted word ax.

2. If two relators overlap non-trivially: $r = abc$, $s = dbf$, we can use s to replace b in r: $r = ad^{-1}f^{-1}c$.
3. If there is a relator in which one generator $x \in \underline{\boldsymbol{g}}$ occurs only once, say $r = axb$, then replace all occurrences of x by $a^{-1}b^{-1}$ and then delete x and r.

In practice (such as the command `SimplifiedFpGroup` in GAP), Tietze transformations perform the following greedy algorithm by repeating the following steps:

1. Eliminate redundant generators using relators of length 1 or 2 (this will not change the total relator length).
2. Eliminate up to n (typically $n = 10$) generators, as long as the total relator length does not grow by more than m ($m = 30\%$).
3. Find common substrings to reduce the total relator length, until the total improvement of a reduction round is less than p ($p = 0.01\%$).

Clearly this has no guarantee whatsoever to produce a "best" presentation, but at least often produces reasonable local minima.

III.3 Algorithms for finitely presented groups

The obvious aim for algorithms would be for example tests for finiteness or computation of group order, however there are some even more basic questions to be resolved first:

In what can be considered the first publication on computational group theory, in 1911 [Deh11] the mathematician Max Dehn asked for algorithms to solve the following problems (called "Dehn Problems" since then):

Word Problem Given a finitely presented group G, is there an algorithm that decides whether a given word represents the identity in G?

Conjugacy Problem Given a finitely presented group G, is there an algorithm that decides whether the elements represented by two words $u, v \in G$ are conjugate in G.

Isomorphism Problem Is there an algorithm that decides whether a pair of finitely presented groups is isomorphic?

These problems have been resolved for some particular classes of presentations. In attacking any such questions in general, however, we are facing an unexpected obstacle, which shows that no such algorithms can exist:

THEOREM III.13 (Boone [Boo57], Novikov [Nov55]): There cannot be an algorithm (in the Turing machine sense) that will test whether any given finitely presented group is trivial.

Proof: Translation to the Halteproblem (stopping problem) for Turing machines. □

Because of this problem the method we present may look strangely toothless, or may be only heuristics. This is a consequence of this fundamental problem.

III.4 Homomorphisms

There is *one* thing that is very easy to do with finitely presented groups, namely working with homomorphisms: We define homomorphisms by prescribing images of the generators. It is easy to test whether such a map is a homomorphism, as long as we can compare elements in the image group:

Lemma III.14: Let $G = \left\langle \underline{g} \mid R \right\rangle$ be a finitely presented group. For a group H we define a map $\varphi: \underline{g} \to H$. Then φ extends to a homomorphism $G \to H$ if and only if for every relator $r \in R$ we have that the relator evaluated in the generator images is trivial: $r(\underline{g}^\varphi) = 1$.

Proof: Homework. □

Clearly evaluating such a homomorphism on an arbitrary element $w(\underline{g})$ simply means evaluating $w(\underline{g}^\varphi)$.

As this test is easy, much of the functionality for finitely presented groups involves homomorphisms – either working with homomorphic images, or finding homomorphisms (so-called "Quotient algorithms"). The easiest of these is probably to find epimorphisms onto a certain group:

Finding Epimorphisms

Given a finitely presented group $G = \left\langle \underline{g} \mid R \right\rangle$ and another (finite) group H, we can find an epimorphism $\varphi: G \to H$ by trying to find suitable images $g_i^\varphi \in H$ for each generator $g_i \in \underline{g}$.

If we have a candidate set of images, they will yield an epimorphism if:

- The relators evaluated in the generator images are trivial: $r(\underline{g}^\varphi) = 1_H$, and
- The generator images generate H: $H = \left\langle \underline{g}^\varphi \right\rangle$ (otherwise we just get a homomorphism.)

As $\underline{g}$ and H are finite, there is just a finite set of generator images to consider for a given H, testing all therefore is a finite process.

If $\varphi: G \to H$ is an epimorphism and $h \in H$, the map $g \mapsto (g^\varphi)^h$ is also an epimorphism, it is the product of φ and the inner automorphism of H induced by h. It therefore makes sense to enumerate images of the generators of G only up to inner automorphisms of H.

Suppose that $\underline{g} = \{g_1, \ldots, g_m\}$ and the images are $\{h_1, \ldots, h_m\}$. If we permit conjugacy by h we can certainly achieve that h_1 is chosen to be a fixed representative in its conjugacy class. This reduces the possible conjugacy to elements of $C_1 = C_H(h_1)$.

Next h_2 can be chosen up to C_1 conjugacy. We can do this by first deciding on the H-class of h_2, say this class has representative r. Then the elements of r^H correspond to $C_H(r)\backslash H$. Thus C_1 orbits on this class correspond to the double cosets $C_H(r)\backslash H/C_1$. Conjugating r by representatives of these double cosets gives the possible candidates for h_2.

We then reduce conjugacy to $C_2 = C_H(h_1, h_2)$ and iterate on h_3.

This yields the following algorithm, called the GQuotient-algorithm (here better: H-quotient) (Holt [HEO05] calls it Epimorphisms):

Algorithm III.15: Given a finitely presented group G and a finite group H, determine all epimorphisms from G to H up to inner automorphisms of H.

Input: $G = \langle g_1, \ldots, g_m \mid R \rangle$
Output: A list L of epimorphisms
begin
1: $L := [\,]$;
2: Let C be a list of conjugacy class representatives for H
3: **for** $h_1 \in C$ **do** {Image of g_1}
4: **for** $r_2 \in C$ **do** {Class of image of g_2}
5: Let D_2 be a set of representatives of $C_H(r_2)\backslash H/C_H(h_1)$.
6: **for** $d_2 \in D_2$ **do** {Image of g_2}
7: $h_2 = r_2^{d_2}$;
8: ...
9: **for** $r_k \in C$ **do** {Class of image of g_k}
10: Let D_k be representatives of $C_H(r_k)\backslash H/C_H(h_1, h_2, \ldots, h_{k-1})$.
11: **for** $d_k \in D_k$ **do** {Image of g_k}
12: $h_k = r_k^{d_k}$;
13: **if** $\forall r \in R$: $r(h_1, \ldots, h_k) = 1$ and $H = \langle h_1, \ldots, h_k \rangle$ **then**
14: Add the map $g_i \mapsto h_i$ to L.
15: **fi**;
16: **od**;
17: **od**;
18: ...
19: **od**;
20: **od**;
21: **od**;
22: return L;
end

Note that this is not completely valid pseudo-code, as (lines 8 and 18) we permit a *variable* number of nested for-loops. In practice this has to be

implemented recursively, or by using a while-loop that increments a list of variables.

Note III.16: Note that the algorithm classifies epimorphisms, not quotient groups. If H has outer automorphisms, we will get several epimorphisms with the same kernel.

Note III.17: If we know that $|G| = |H|$ we will in fact find isomorphisms between G and H. In fact, if G and H are both permutation groups, once we determine a set of defining relators for G (section III.10) this approach offers a naive isomorphism test. In such a situation more restrictions on the generator images become available and help to reduce the search space.

If we set $G = H$ and run through all possibilities, we find automorphisms of G up to inner automorphisms and thus can determine generators for $\mathrm{Aut}(G)$.

There are newer and better algorithms for isomorphism and automorphism group of permutation groups.

III.5 Quotient subgroups

Staying with the homomorphism paradigm, the most convenient way to represent arbitrary subgroups of finite index is as pre-images under a homomorphism.

Definition III.18: Let G be a finitely presented group. A *quotient subgroup* (φ, U) of G is a subgroup $S \leq G$ that is given as preimage $S = \varphi^{-1}(U)$ of a subgroup $U \leq G^\varphi$ where $\varphi\colon G \to H$ is a homomorphism into a (typically finite) group.

The idea behind quotient subgroups is that we can calculate or test properties in the image, thus reducing for example to the case of permutation groups. For example:

- $g \in S$ if and only if $g^\varphi \in U = S^\varphi$.
- The quotient representation for $N_G(S)$ is $(\varphi, N_{G^\varphi}(U))$.
- The core (intersection of conjugates) of S is $(\varphi, \mathrm{Core}_G(U))$.
- If $S, T \leq G$ are both quotient subgroups given by the homomorphisms φ and ψ respectively, we can consider the larger quotient $\xi\colon G \to G^\varphi \perp G^\psi$ and calculate the intersection there.

If we have a quotient subgroup $S = \varphi^{-1}(U) \leq G$ the cosets $S\backslash G$ are in bijection with the cosets $U\backslash G^\varphi$. We can thus compare cosets or consider the action of G on the cosets of S. As S is the point stabilizer in this action, Schreier generators for this give a set of generators for S, thus converting a quotient subgroup in a "traditional" subgroup given by generators.

III.6 Coset Enumeration

In practice, we will often have subgroups given by generators and not as a quotient subgroup. Coset enumeration is a method that will produce the permutation representation on the cosets of this subgroup, provided it has finite index. This representation is an obvious choice to represent the subgroup as a quotient subgroup.

The fundamental idea behind the algorithm is that we perform an orbit algorithm on the cosets of the subgroup. As we do not have a proper element test we might not realize that certain cosets are the same, but we can eventually discover this using the defining relators of the group.

The method, one of the oldest group theoretic procedures, was originally proposed for hand calculations. It is often named after the inventors as the "Todd-Coxeter algorithm" or simply as "Coset Enumeration".

Note III.19: In view of theorem III.13 the term "algorithm" is problematic (and ought to be replaced by "method"): The runtime cannot be bounded, that is the calculation may not finish in any preset finite time.

The main tool for coset enumeration is the *coset table*. It lists the cosets of the subgroup and for each coset the images under every generator and generator inverse. Coset 1 is defined to be the subgroup. Every other coset is defined to be the image of a prior coset under a generator.

We also maintain a table for every relator. These tables trace the images of every coset under the relator generator by generator. We know (as the relator has to be trivial in the group) that the coset needs to remain fixed.

Finally we keep a similar table for every subgroup generator, here however we only require the trivial coset to remain fixed.

Example III.20: Consider $G = \langle a, b \mid a^2 = b^3 = (ab)^5 = 1\rangle$ and $S = \langle a, a^b\rangle \leq G$. Then the coset table is

	a	a^{-1}	b	b^{-1}
1				

the relator tables are:

a	a
1	1

b	b	b
1		1

a	$babababa$	b
1		1

and the subgroup tables are:

a	
1	1

b^{-1}	a	b
1		1

We now start defining new cosets by taking the image of an existing coset under a generator or its image. We enter the inverse information: If $a^g = b$ then $b^{g^{-1}} = a$. We also fill in all entries in the tables whose images become defined.

What may happen, is that such an entry fills the last hole in a table. Then we get an *deduction* that the image of one coset under the next generator must be the

existing value on the other side of the table entry. We enter these deductions in the table as if they were definitions.

Example III.21 (Continued): In our example the first subgroup table immediately tells us that $1^a = 1$. We enter this in the coset table. (We will use underlines to denote definitions of cosets and bold numbers to denote the most recent change. An exclamation mark denotes a deduction.)

	a	a^{-1}	b	b^{-1}
1	**1**		**1**	

We also update the other tables:

$a\ a$
1 **1**!1

bbb
1 1

$a\ bababababab$
1 **1** 1

a
1 1

$b^{-1}ab$
1 1

We get a deduction $1^a = 1$, but this happens to be not new.

Because the subgroup tables carry only one row (for the trivial coset) we will retire the table for the generator a from now on.

Next we define coset 2 to be the image of coset 1 under b:

	a	a^{-1}	b	b^{-1}
1	**1**	1	**2**	
2				**1**

$a\ a$
1 1 1
2 2

$b\ b\ b$
1 **2** 1
2 **1** 2

$a\ b\ ababab\ a\ b$
1 1 **2** 1
2 **1 1** 2

$b^{-1}ab$
1 1

Define $3 = 1^{b^{-1}}$:

	a	a^{-1}	b	b^{-1}
1	1	1	2	**3**
2				1
3			**1**	

$a\ a$
1 1 1
2 2
3 3

$b\ b\ b$
1 2!**3** 1
2!**3** 1 2
3 **1** **2**!3

$a\ b\ ababa\ b\ a\ b$
1 1 2 **3** 1
2 **3** 1 1 2
3 3

$b^{-1}\ a\ b$
1 **3**!**3** 1

We conclude that $2^b = 3$ and $3^a = 3$ (and now also retire the second subgroup table).

	a	a^{-1}	b	b^{-1}
1	1	1	2	3
2			**3**	1
3	**3**	**3**	1	**2**

$a\ a$
1 1 1
2 2
3 **3**!3

$b\ b\ b$
1 2 3 1
2 3 1 2
3 1 2 3

$a\ b\ a\ b\ a\ b\ a\ b\ a\ b$
1 1 2 **2 3** 3 1
2 **2 3** 3 1 1 2
3 **3 1 1 2** **2** 3

There is no new conclusion. We set $2^a = 4$

	a	a^{-1}	b	b^{-1}
1	1	1	2	3
2	**4**		3	1
3	3	3	1	2
4		**2**		

$a\ a$
1 1 1
2 **4**!2
3 3 3
4!**2** 4

$b\ b\ b$
1 2 3 1
2 3 1 2
3 1 2 3
4 4

$a\ b\ a\ b\ a\ b\ a\ b\ a\ b$
1 1 2 **4** 2 3 3 1
2 **4** 2 3 3 1 1 2
3 3 1 1 2 **4** 2 3
4 4

We conclude $4^a = 2$

	a	a^{-1}	b	b^{-1}
1	1	1	2	3
2	4	**4**	3	1
3	3	3	1	2
4	**2**	2		

$a\ a$
1 1 1
2 4 2
3 3 3
4 2 4

$b\ b\ b$
1 2 3 1
2 3 1 2
3 1 2 3
4 4

$a\ b\ a\ b\ a\ b\ a\ b\ a\ b$
1 1 2 4 **4** 2 3 3 1
2 4 **4** 2 3 3 1 1 2
3 3 1 1 2 4 **4** 2 3
4 **2 3 3 1 1 2 4** 4

Now we set $4^b = 5$:

	a	a^{-1}	b	b^{-1}
1	1	1	2	3
2	4	4	3	1
3	3	3	1	2
4	2	2	**5**	
5				**4**

$a\ a$
1 1 1
2 4 2
3 3 3
4 2 4
5 5

$b\ b\ b$
1 2 3 1
2 3 1 2
3 1 2 3
4 **5** 4
5 **4** 5

$a\ b\ a\ b\ a\ b\ a\ b\ a\ b$
1 1 2 4 **5** 4 2 3 3 1
2 4 **5** 4 2 3 3 1 1 2
3 3 1 1 2 4 **5** 4 2 3
4 2 3 3 1 1 2 4 **5** 4
5 **4** **4** 5

As there is no deduction, we define $4^{b^{-1}} = 6$.

	a	a^{-1}	b	b^{-1}
1	1	1	2	3
2	4	4	3	1
3	3	3	1	2
4	2	2	5	**6**
5				4
6			**4**	

$a\ a$
1 1 1
2 4 2
3 3 3
4 2 4
5 5
6 6

$b\ b\ b$
1 2 3 1
2 3 1 2
3 1 2 3
4 5!**6** 4
5!**6** 4 5
6 **4** **5**!6

$a\ b\ a\ b\ a\ b\ a\ b\ a\ b$
1 1 2 4 5!**6** 4 2 3 3 1
2 4 5!**6** 4 2 3 3 1 1 2
3 3 1 1 2 4 5!**6** 4 2 3
4 2 3 3 1 1 2 4 5!**6** 4
5 **6** 4 4 5
6 6

We conclude $5^a = 6$ and $5^b = 6$. The second table then implies $6^a = 5$. (Also all relator tables are filled with no new deduction.)

	a	a^{-1}	b	b^{-1}
1	1	1	2	3
2	4	4	3	1
3	3	3	1	2
4	2	2	5	6
5	**6**	**6**	**6**	4
6	**5**	**5**	4	**5**

At this point all places in the table are closed and no deductions pending.

Once we have reached the point of all tables closed and no deductions pending, the columns of the coset table give permutation images for the group generators that are consistent with all relators (as we maintained the relator tables).

If there are n rows, we have thus obtained a homomorphism $\varphi: G \to S_n$, such that $S^\varphi = \mathrm{Stab}_{G^\varphi}(1)$. This is all we need to represent S as a quotient subgroup. (In particular, $\left[G:S\right] = n$ equals the number of rows in the table.)

In the example we would have $a \mapsto (2,4)(5,6)$ and $b \mapsto (1,2,3)(4,5,6)$. We can also read off coset representatives as follows:

$$\begin{aligned} 2 &= 1^b \\ 3 &= 1^{b^{-1}} \\ 4 &= 1^{ba} \\ 5 &= 1^{bab} \\ 6 &= 1^{bab^{-1}} \end{aligned}$$

Note III.22: On the computer, we can save space by not storing images under inverses and the subgroup and relator tables – we can simply compute their contents by *scanning* through relators (i.e. looking at images of cosets under subsequent generators within the relator *forwards and backwards*), respectively by looking up images.

To avoid having to scan through all relators and all cosets, the following observation is useful: After a definition (or deduction) occurs obviously only relators that make use of this new definition are of interest for renewed checking. Suppose that $abcd$ is a relator, which scans at coset x only up to ab but hangs at c (ignoring the scan from the right). Then a new result can occur only if the coset x^{ab} (meaning the coset if we apply ab to coset x) gets its image under c defined.

In this situation we can instead consider the (equivalent) relator $abcd^{ab} = cdab$ and consider it being scanned starting at coset x^{ab}.

We therefore perform the following preprocessing: We form a list of all cyclic permutations of all relators and their inverses and store these according to the first letter occurring in the permuted relator. Let R_g^c be the set of all such permutations that start with the letter g.

Then if the image of coset y under generator g is defined as z, we scan all relators in R_g^c starting at the coset y and all relators in $R_{g^{-1}}^c$ starting at $z = y^g$.

This then will take care of any relator that might have scanned partially before and will scan further now.

Coincidences

There is one other event that can happen during coset enumeration. Closing a table row might imply the equality of two (prior considered different) cosets. In this case we will identify these cosets, deleting one from the tables. In doing this identification, (partially) filled rows of the coset table might imply further coincidences. We

thus keep a list of coincidences to process and add to it any such further identifications and process them one by one.

Example III.23: For the same groups as in the previous example, suppose we would have followed a different definition sequence, and doing so ended up with the following tables. (The underlined numbers indicate the definition sequence.)

	a	a^{-1}	b	b^{-1}
1	1	1	$\underline{2}$	
2	3	3		1
3	2	2	$\underline{4}$	$\underline{5}$
4	$\underline{6}$	6	5	3
5			3	4
6	4	4		

a	a
1	1
2	3
3	2
4	6
5	5
6	4

b	b	b
1	2	1
2	1	2
3	4	5
4	5	3
5	3	4
6	6	6

a	b	a	b	a	b	a	b	a	b
1	1	2	3	4	6				1
2	3	4	6				1	1	2
3	2							5	3
4	6					1	1	2	3	4
5							6	4	5
6	4	5							6

$b^{-1}ab$
1 1

We now define $1^{b^{-1}} = 7$ and get (from $b^3 : 7 \to 1 \to 2 \to 7$) that also $2^b = 7$. Furthermore, it lets us fill the second subgroup table:

b^{-1}	a	b
1	7!7	1

We get the consequence $7^a = 7$ and similarly $7^{a^{-1}} = 7$. Thus we get

	a	a^{-1}	b	b^{-1}
1	1	1	$\underline{2}$	$\underline{7}$
2	3	3	7	1
3	2	2	$\underline{4}$	$\underline{5}$
4	$\underline{6}$	6	5	3
5			3	4
6	4	4		
7	7	7	1	2

a	a
1	1
2	3
3	2
4	6
5	5
6	4
7	7

b	b	b
1	2	7	1
2	7	1	2
3	4	5	3
4	5	3	4
5	3	4	5
6			6
7	1	2	7

a	b	a	b	a	b	a	b	a	b
1	1	2	3	4	6	3	2	7	7	1
2	3	4	6	3	2	7	7	1	1	2
3	2	7	7	1	1	2	3	4	5	3
4	6	3	2	7	7	1	1	2	3	4
5								6	4	5
6	4	5								6
7	7	1	1	2	3	4	5	3	2	7

with the implications $6^b = 3$ and $4^a = 5$. As we had $4^a = 6$, cosets 5 and 6 must coincide. (As we had $6^b = 3$ and $5^b = 3$ there is no subsequent coincidence.)

After this coagulation the table is again closed:

	a	a^{-1}	b	b^{-1}
1	1	1	2	7
2	3	3	7	1
3	2	2	4	5
4	5	5	5	3
5	4	3	3	4
7	7	7	1	2

Strategies

As the examples show, the performance of coset enumeration depends crucially on the definition sequence (i.e. which cosets are defined as what images at what point). A large body of literature exists that outlines experiments and strategies. The two main strategies, named after their initial proposers are:

Felsch (1959/60) Define the next coset as the first open entry (by rows and within a row by columns) in the coset table. This guarantees that the image of each coset under each generator will be defined at some point.

HLT (1953, for Haselgrove[1], Leech and Trotter). If there are gaps of length 1 in a subgroup or relator table, fill these gaps (in the hope of getting immediately a consequence). This method is harder to understand theoretically, but often performs better in practice.

There is a large corpus of variants and modifications to these strategies (for example the addition of redundant relators). In particular with hard enumerations often just particular variants will finish.

THEOREM III.24 (Mendelsohn, 1964): Suppose that $\left[G{:}S\right] < \infty$ and that the strategy used guarantees that for every defined coset a and every generator g the images a^g, and $a^{g^{-1}}$ will be defined after finitely many steps, then the coset enumeration terminates after finitely (but not bounded!) many steps with the correct index.

Proof:(Idea) If the process terminates, the columns yield valid permutations for the action on the cosets of S. To show termination, assume the contrary. By the condition we can (by transfinite induction) build an *infinite* coset table which would contradict $\left[G{:}S\right] < \infty$. □

Applications and Variations

A principal use of coset enumeration is to get a quotient representation for subgroups for purposes such as element tests or subgroup intersection. We will also see in section III.8 that the coset tables themselves find use in the calculation of subgroup presentations.

[1] Indeed with "s", not with "z"

One obvious application is the size of a group, by enumerating the cosets of the trivial subgroup. (However in practice one enumerates modulo a cyclic subgroup and obtains a subgroup presentation.)

In general there are many different coset tables corresponding to one subgroup which simply differ by the labels given to the different cosets. For comparing coset tables or processing them further it can be convenient to relabel the cosets to bring the table into a "canonical" form.

DEFINITION III.25: A coset table is *standardized* if when running through the cosets and within each coset through the generator images (ignoring generator inverses), the cosets appear in order of the integers $1, 2, 3, \ldots$.

A standardized coset table thus is the coset table we would obtain if we performed a pure Felsch-style enumeration and after each coincidence relabeled cosets to avoid "gaps".

If we have a coset table we can easily bring it into standard form by running through cosets and within cosets through generator images and reassigning new labels according to the order in which cosets appear.

The following lemma now is obvious:

LEMMA III.26: There is a bijection between subgroups of G of index n and standardized coset tables for G with n cosets.

III.7 Low Index Subgroups

A prominent variation of coset enumeration is the so-called *Low-Index* algorithm that for a given n will find all subgroups of a finitely presented group $G = \left\langle \underline{g} | R \right\rangle$ of index $\leq n$ (up to conjugacy).

We will construct these subgroups by constructing all valid standardized coset tables for G on up to n cosets.

For simplicity let us initially assume that we do not want to eliminate conjugates:

The basic step in the algorithm (computing one descendant) takes a partially completed, standardized coset table, involving $k \leq n$ cosets. (The initialization is with the empty coset table.) If the table is in fact complete, it yields a subgroup.

Otherwise we take the next (within cosets and within each coset in order of generators) open definition, say the image of coset x under generator g.

We now split up in several possibilities on assigning this image: We can assign x^g to be one of the existing cosets $1, \ldots, k$, or (if $k < n$) a new coset $k+1$.

For each choice we take a copy of the coset table and make in this copy the corresponding assignment. Next we run the deduction check, as in ordinary coset enumeration. (According to remark III.22, we only need to scan the relators in R_g^c at coset x and $R_{g^{-1}}^c$ at x^g, as well as relators for consequential deductions.) We enter

deductions in the table. However if a coincidence occurs, we know that we made an invalid choice, and abandon this partial table, backtracking to the next (prior) choice.

Otherwise we take this new partial table (which so far fulfills all relators as far as possible and is standardized by the way we selected the next open definition) and compute its further descendants.

More formally, this gives the following algorithm:

ALGORITHM III.27: This is a basic version of the low index algorithm without elimination of conjugates.

Input: $G = \langle \underline{G} \mid R \rangle$, index n
Output: All subgroups of G of index up to n, given by coset tables.
begin
Initialize T as empty table for G.
$L := []$;
return DESCENDANTS(T).
end

The DESCENDANTS routine performs the actual assignment and calls a second routine TRY to verify validity and process deductions. We assume that an image 0 indicates that the image is not yet defined.
DESCENDANTS(T)
begin
1: **if** T is complete **then**
2: Add T to L;
3: **else**
4: m=number of cosets defined in T
5: Let coset x under generator g be the first undefined ($x^g = 0$) image.
6: **for** $y \in [1..m]$ **do**
7: **if** $y^{g^{-1}} = 0$ **then** {otherwise we have an image clash}
8: Let S be a copy of T;
9: In S set $x^g = y$ and $y^{g^{-1}} = x$;
10: TRY(S,x,g);
11: **fi**;
12: **od**;
13: **if** $m < n$ **then** {is one more coset possible?}
14: Let S be a copy of T
15: Add coset $m+1$ to S;;
16: In S set $x^g = m+1$ and $(m+1)^{g^{-1}} = x$;
17: TRY(S,x,g);
18: **fi**;
19: **fi**;
end

The validity test is the last routine. It takes a partial coset table S in which

an assignment x^g has just been made and then performs dependencies and continue search if no coincidences arise.
TRY(S, x, g)
begin
1: Empty the deduction stack;
2: Push x, g on the deduction stack;
3: Process deductions for S as in coset enumeration;
4: **if** no coincidence arose **then**
5: call DESCENDANTS(S);
6: **fi**;
end

Next we want to consider conjugacy. Using a standard approach to the construction of objects up to a group action, we define a (somehow arbitrary) order on coset tables which tests conveniently for partial coset tables, and then for each (partial) table test whether it can be the smallest in its class (the group acting by conjugation of subgroups in our case). If not we discard the candidate.

Such a test would be performed before line 5 of the function TRY.

The ordering of coset tables we use is lexicographic, considering the table row by row. I.e. for two tables S, T of size n we have that $T < S$ if for some $1 \le x \le n$ and some generator g the following holds:

- For all $y < x$ and any generator h, we have that y^h is the same in S and T.
- For all generators h before g we have that x^h is the same in S and T.
- x^g is smaller in T than in S.

To determine the coset table for a conjugate, observe that a coset table yields the conjugation action on the cosets of a subgroup. In this action the subgroup is the stabilizer of the point 1, and every conjugate is the stabilizer of another point x. If $g \in G$ is an element such that $1^g = x$, then g would conjugate the subgroup to the conjugate corresponding to x. Among coset tables, this conjugation would happen as relabeling of points and permutation of cosets by g.

But (the permutation action *is* given by the coset table) we can determine such an element g from the coset table.

As we have only a partial coset table this construction may not yet succeed (or we may be lacking entries to compare yet), in any case it will eliminate groups that are not first in their class. We also often can perform this pruning already for an only partially constructed coset table.

PERFORMANCE III.28: There are many variations and improvements. For example, as long relators rarely yield a deduction but only are conditions to test, it can make sense to only consider the shorter (whatever this means) relators for the determination of coset tables and simply test each table obtained afterwards for the remaining relators.

Note III.29: There are variations to only obtain normal subgroups. However, given the knowledge of all small groups up to order 2000, the following approach makes more sense: If $N \lhd G$ has small index consider $Q = G^{\varphi} = G/N$.

Typically either Q is solvable or even nilpotent (and then N can be found via powerful quotient algorithms) or Q has a faithful permutation representation on the cosets of a subgroup $U \leq Q$ of small index. (Here we use the knowledge of groups of small order to obtain concrete bounds.)

Then the preimage of U under φ can be obtained by an ordinary low-index calculation for such a small index.

An somewhat alternative view of this is to consider the low-index algorithm as a version of the GQuotient algorithm III.15. Enumerating all possible columns of the coset table is in effect like enumerating all m-tuples of elements in the symmetric group S_n that fulfill the relations, replacing the "surjectivity" condition to be just transitivity of the image. The principal benefit of the low-index routine is that it implicitly uses the defining relators to impose conditions on the permutations.

III.8 Subgroup Presentations

Any subgroup of finite index of a finitely presented group is finitely generated by lemma I.16. In fact it also is finitely presented:

To state the theorem we need some definitions: Let $F = \langle f_1, \ldots, f_m \rangle$ a free group and $R = \{r_1(\underline{f}), \ldots, r_k(\underline{f})\}$ a finite set of relators that defines the finitely presented group $G = \left\langle \underline{g} \mid R \right\rangle$ as a quotient of F. We consider $\underline{g}$ as the elements of G that are the images of the free generators $\underline{f}$.

Suppose that $S \leq G$ with $n = \left[G{:}S\right] < \infty$. We choose a transversal of coset representatives for S: $t_1 = 1, t_2, \ldots, t_{\left[G:S\right]}$, and form the Schreier generators (lemma I.16) $s_{i,j} = t_i g_j (\overline{t_i g_j})^{-1}$ for S.

If the coset representative t_i is defined as $t_i = t_j \cdot g_x$, the Schreier generator $s_{j,x}$ is trivial by definition. We call the pair (j,x) "redundant" and let $I \subset \{1,\ldots,n\} \times \{1,\ldots,m\}$ be the set of all index pairs that are not redundant, i.e. the set of Schreier generators that are not trivial by definition is $\{s_{i,j} \mid (i,j) \in I\}$. As there are $n-1$ coset representatives that are defined as image of a "smaller" coset representative under a group generator, we have $|I| = n \cdot m - (n-1) = n \cdot (m-1) + 1$.

As G is a quotient of F, we have a subgroup $U \leq F$ which is the full preimage of S under the natural epimorphism $F \to G$.

Now consider that $w(f_1, \ldots, f_m) \in U$ is a word in $\underline{f}$ such that the corresponding element $w(g_1, \ldots, g_m) \in S$. Then we can (as in the proof of Schreier's theorem I.16) rewrite $w(g_1, \ldots, g_m)$ as a word in the Schreier generators $s_{i,j}$. (For this we only need to know the action of G on the cosets of S.)

We form a second free group E on a generating set $\{e_{i,j} \mid (i,j) \in I\}$. Let $\rho: U \to E$ be the *rewriting map*, which for any such word $w(f_1, \ldots, f_m) \in U$ returns a word $\rho(w) \in E$ which represents the expression of $w(g_1, \ldots, g_m)$ as a word in the nonredundant Schreier generators $s_{i,j}$.

Theorem III.30 (Reidemeister): Let $G = \left\langle \underline{g} \mid R \right\rangle$ a finitely presented group and $S \leq G$ with $\left[G{:}S\right] < \infty$. Then S is finitely presented and

$$S = \left\langle e_{i,j} \text{ for } (i,j) \in I \mid \rho(t_x r_y t_x^{-1}), 1 \leq x \leq n, 1 \leq y \leq k \right\rangle$$

is a presentation for S on the Schreier generators. (We are slightly sloppy in the notation here by interpreting the t_x as representatives in F.)

Proof: If we evaluate $t_x^{-1} r_y t_x$ in the generators $\underline{g}$ of G, these relators all evaluate to the identity. Therefore the rewritten relators must evaluate to the identity in S. This shows that there is an epimorphism from the finitely presented group onto S.

We thus only need to show that any relation among the $s_{i,j}$ can be deduced from the rewritten relators: Let $w(\underline{e})$ be a word such that $w(\underline{s}) = 1$ in S. By replacing $e_{i,j}$ by $(t_i f_j \overline{t_i f_j})^{-1}$ we can get this as a new word $v(\underline{f})$, such that $v(\underline{g}) = 1$ in G. Therefore v can be expressed as a product of conjugates of elements in R: $v(\underline{f}) = \prod_z r_{x_z}^{u_z(\underline{f})}$ where the u_z denote words for the conjugating elements.

Now consider a single factor $r^{u(\underline{f})}$. As the t_x are representatives for the right cosets of S, we can write $u(\underline{f}) = t_x^{-1} \cdot q(\underline{f})$ where $q(\underline{g}) \in S$ and x is defined by $S \cdot u(\underline{g})^{-1} = S \cdot t_x$. Thus $r^{u(\underline{f})} = (t_x \cdot r \cdot t_x^{-1})^{q(\underline{f})}$. Rewriting this with ρ, we get $\rho(t_x \cdot r \cdot t_x^{-1})^{\rho(q)}$, which is a conjugate of a rewritten relator. □

We note an important consequence:

Corollary III.31 (Nielsen, Schreier): Any subgroup of finite index n of a free group of rank m is free of rank $n \cdot (m-1) + 1$.

Proof: Rewriting will produce no relators for the subgroup. □

Note III.32: This theorem also holds without the "finite index" qualifier. It is usually proven in this general form using algebraic topology (coverings).

Note III.33: Every generating set of a free group must contain at least as many elements as its rank. (Proof: Consider the largest elementary abelian quotient that is a 2-group $Q = F/F'F^2$. The images of a generating set generate Q as vector space, the rank of F is the dimension of Q. Then use elementary linear algebra.) This proves that the number of Schreier generators given by lemma I.16 in chapter I cannot be improved on in general.

To perform this rewriting in practice, it is easiest to form an *augmented coset table* by storing (for entries that are not definitions) in position for coset x and generator g also the appropriate Schreier generator s, such that $t_x \cdot g = s \cdot t_x$. We can

construct this from the permutation action on the cosets by a simple orbit algorithm.

We then scan every relator at every coset and collect the occurring Schreier generators.

Note III.34: In practice, Reidemeister rewriting often produces many trivial Schreier generators and relators that are trivial or of length 1 or 2 (which immediately eliminate generators). Thus typically the resulting presentation is processed by Tietze transformations to eliminate such trivialities.

Example III.35: Let us go back to example III.20 where we enumerated cosets. Our coset table was

	a	a^{-1}	b	b^{-1}
1	1	1	$\underline{2}$	$\underline{3}$
2	$\underline{4}$	4	3	1
3	3	3	1	2
4	2	2	$\underline{5}$	$\underline{6}$
5	6	6	6	4
6	5	5	4	5

with representatives

$$\begin{aligned} t_1 &= 1 \\ t_2 &= b \\ t_3 &= b^{-1} \\ t_4 &= ba \\ t_5 &= bab \\ t_6 &= bab^{-1} \end{aligned}$$

This defines the following nontrivial Schreier generators and the augmented coset table:

$$\begin{aligned} c &= t_1 a t_1^{-1} = a \\ d &= t_2 b t_3^{-1} = b^3 \\ e &= t_3 a t_3^{-1} = b^{-1}ab \\ f &= t_4 a t_2^{-1} = baab^{-1} \\ g &= t_5 a t_6^{-1} = bababa^{-1}b^{-1} \\ h &= t_5 b t_6^{-1} = babbba^{-1}b^{-1} \\ i &= t_6 a t_5^{-1} = bab^{-1}ab^{-1}a^{-1}b^{-1} \end{aligned}$$

and

	a	a^{-1}	b	b^{-1}
1	$c1$	$c^{-1}1$	$\underline{2}$	$\underline{3}$
2	$\underline{4}$	$f^{-1}4$	$d3$	1
3	$e3$	$e^{-1}3$	1	$d^{-1}2$
4	$f2$	2	$\underline{5}$	$\underline{6}$
5	$g6$	$i^{-1}6$	$h6$	4
6	$i5$	$g^{-1}5$	4	$h^{-1}5$

Now we trace the relators at every coset and collect the Schreier generators on the way:

	a^2	b^3	$(ab)^5$
1	c^2	d	$cgfde$
2	f	d	$gfdec$
3	e^2	d	$ecgfd$
4	f	h	$fdecg$
5	gi	h	$gfdec$
6	ig	h	$(ih)^5$

Eliminating duplicates and cyclic permutations (which are just conjugates) we get the presentation

$$S = \left\langle c, d, e, f, g, h, i \mid c^2 = d = e^2 = f = h = gi = ig = cgfde = (ih)^5 = 1 \right\rangle$$

We eliminate trivial and redundant ($i = g^{-1}$) generators and get

$$S = \left\langle c, e, g \mid c^2 = e^2 = cge = (g^{-1})^5 = 1 \right\rangle$$

We now can eliminate $g = c^{-1}e^{-1}$ and get

$$S = \left\langle c, e \mid c^2 = e^2 = (ec)^5 = 1 \right\rangle$$

which is easily seen to be a dihedral group of order 10 (so the initial group G must have had order $6 \cdot 10 = 60$).

Note III.36: The occurrence of cyclic conjugates of the relator $cgfde$ is not really surprising, but simply a consequence of the power relator $(ab)^5$. One can incorporate this directly in the rewriting algorithm, similarly to a treatment (generator elimination) for relators of length 1.

Note III.37: A small variant of this rewriting process is the so-called *Modified Todd-Coxeter* algorithm that produces a presentation for S in a given generator set. In general it produces worse presentations than the presentation in Schreier generators obtained here.

As an application we describe a method often used to determine the order of a finite finitely presented group: We enumerate the cosets of a cyclic subgroup $\langle x \rangle$ (often x is itself chosen as a generator of the group). Then we rewrite the presentation to obtain a presentation for $\langle x \rangle$. Then $|G| = \left[G\colon \langle x \rangle\right] \cdot |\langle x \rangle|$. Since the subgroup is known to be cyclic, any resulting relator will in effect have the form $x^{e_i} = 1$, thus $|\langle x \rangle| = \operatorname{lcm}_i(e_i)$ can be obtained easily.

Performance III.38: As the rewritten presentation is on $n(m-1)+1$ generators, even Tietze transformations typically cannot rescue humongous presentations obtained for subgroups of large index. Thus there is a natural limit (a few thousand) on the subgroup index for which rewriting is feasible.

III.9 Abelian Quotients

We have seen already a method (the GQuotient algorithm, algorithm III.15) which for a finitely presented group G finds all quotient groups G/N isomorphic to a given finitely presented group H.

In general, one would like to do this not only for a specific H, but for the "largest possible H" within some class of groups. Such algorithms are called "quotient algorithms", we will encounter them again later in section IV.5.

Here we want to determine the largest abelian quotient. By the "quotient subgroup" paradigm, this is equivalent to determining the derived subgroup G' of a finitely presented group $G = \left\langle \underline{g} \mid R \right\rangle$.

The principal idea is the following observation: Suppose that F is the free group in which the presentation for G is given and $\varphi\colon F \to G$ is the epimorphism. Let $N = F' = \left\langle x^{-1}y^{-1}xy \mid x, y \in F \right\rangle_F$. then $N^\varphi = G'$. Thus $F/N \cdot \operatorname{Kern}\varphi \cong G/G'$.

We thus get a description for G/G' by simply *abelianizing* the presentation for G, i.e. considering it as a presentation for an abelian group.

As the generators in an abelian group commute, we can describe the relators for G/G' by a matrix A: The columns correspond to the generators of G/G' (images of the generators of G), each row represents one relator, which we can assume to be in the form $g_1^{e_1} g_2^{e_2} \cdots g_m^{e_m}$, we simply store the exponent vector $[e_1, \ldots, e_m]$.

Now consider the effect of elementary transformations over $\mathbb{Z}$ on the rows and columns of A (i.e. swap, adding a multiple of one to another and multiplication by ± 1): Such transformations on the rows correspond to a change of the relator set R, but (as they are invertible) these new relators will generate the same group. Transformations of the columns correspond to a generator change for G/G'. Again the invertibility of the transformation shows that the new elements still generate the whole of G/G'.

We now recall, that we can use such transformations to compute the *Smith Normal Form* of A, i.e. we can transform A into a diagonal matrix S with divisibility conditions among the diagonal entries[2].

This new matrix S will describe a group isomorphic to G/G'. As S is diagonal, this group is simply a direct product of cyclic groups of orders given by the diagonal entries (using order ∞ for entry 0).

If we do not only compute the Smith Normal Form S of A, but also determine matrices $A = P \cdot S \cdot Q$, the matrix Q describes the necessary change of the generating system, thus Q^{-1} describes how to form a homomorphism $G \to C$ with $C \cong G/G'$ the abelian group given by the diagonal entries in S.

Performance III.39: The bottleneck of such a calculation is that — even if the entries in S are small — the calculation of the Smith Normal Form can often produce intermediate coefficient explosion. (It becomes even worse for the (non-unique!) transformation matrices P and Q.) There is an extensive literature considering strategies for such calculations (in particular on how to keep entries in P and Q small).

To indicate the difficulty, note that the standard approach of reduction modulo a prime does not work, because we can always scale modulo a prime. One way to rescue this is to use a theorem relating the diagonal entries of S to the gcd's of determinants of minors of A, and calculating these determinants modulo a prime. This however does not yield transforming matrices.

Abelianized rewriting

We can combine the algorithms of this section and the previous one and ask for the abelian invariants of a subgroup. Instead of performing one algorithm after the other, rewriting relators and then abelianizing them, it is beneficial to immediately abelianize relators while rewriting. This avoids maintaining long relators intermediately and leads to a much more nimble performance.

[2] In fact we only need diagonalization here (which is not a unique form).

Determining the abelianization of a subgroup is one of a handful methods known for determining the infinity of certain groups (there is no universal method): Find a subgroup (of smallish index) whose abelian quotient is infinite.

III.10 Getting a Presentation for a permutation group

In some situations we have a group G already given as a permutation group, but want to obtain a presentation for G. This occurs for example when computing complements to a normal subgroup (see IV.4) or to test whether a map on generators extends to a homomorphism.

In GAP such functionality is provided by the following commands:

`IsomorphismFpGroup` lets GAP choose the generating system in which the presentation is written (typically yielding more generators but a nicer presentation). `IsomorphismFpGroupByGenerators` produces a presentation in a particular generating system.

Reverse Todd-Coxeter

A basic algorithm is due to [Can73], it might be considered easiest as a reversal of coset enumeration:

Suppose we start a coset enumeration for G acting on the cosets of the trivial subgroup, starting without relators. We write t_x to denote the representative for coset x as given by the coset table.

At some point we will define a new coset y which is in fact equal to an already existing coset x. Thus $t_x t_y^{-1}$ must be trivial in G and thus must be a relator. We add this as a relator to the enumeration process (and fill in the corresponding relator table as far as possible). We continue with this until we get and up with a complete table.

Clearly we only added valid relators for G. On the other hand these relators define a group which (as we can see by performing a coset enumeration by the trivial subgroup) has the same order as G, thus the relators yield a presentation for G.

In a variation, suppose that $S \leq G$ and that we know already a presentation of S. We now form a presentation on the generators for S together with the generators for G. As relators we start with the known relators for S as well as relators that express the generators for S as words in the generators for G. Then start a coset enumeration for the cosets of S in G. If two cosets x and y seem different we know that $t_x t_y^{-1} \in S$, thus we can express it as a word in the generators of S. The corresponding relator would have enforced equality of x and y and thus is added to the set of relators.

By the same argument as before, the result will be a presentation for G. We can use Tietze-transformations to eliminate the generators of S and obtain a presentation purely in the generators of G though typically of longer total relator length.

We can iterate this process over a chain of subgroups. In particular we can do this for the subgroups in a stabilizer chain and get a presentation in a strong gen-

erating set.

An application of this is a test whether a map from a permutation group to another group, given by generator images, extends in fact to a homomorphism. Construct a stabilizer chain for the prospective homomorphism. Then proceed as if constructing a presentation. Instead of adding relators, check whether the relators evaluate trivially in the generator images.

NOTE III.40: We can use this method as well, if we want to verify a stabilizer chain that has been obtained with random methods, and might indicate the group being too small: Using this chain, we compute a presentation and then check that the group generators fulfill this presentation. If the chain was too small they will not. This yields the so-called "Todd-Coxeter-Schreier-Sims" algorithm mentioned in section II.1.

Despite the fact that the Todd-Coxeter method has no bounds on the runtime whatsoever, this produces a respectable performance in practice. (See also section III.10.

Using the extension structure

The presentations obtained with this method often are rather messy. It therefore often makes sense to use more information about the composition structure of G and to build a presentation for G from presentations for its composition factors.

The cost of this is that we get a presentation in a new generating set. In most applications this is of little concern.

The heart of this method is the following easy lemma:

LEMMA III.41: Let $N = \langle \underline{\boldsymbol{n}} \rangle \lhd G$ and $G = \left\langle N, \underline{\boldsymbol{g}} \right\rangle$. Suppose that $N = \langle \underline{\boldsymbol{m}} \mid R_1 \rangle$ is a presentation for N and that $G/N = \langle \underline{\boldsymbol{h}} \mid R_2 \rangle$ is a presentation for G/N such that $h_i = Ng_i$. For an element $x \in N$ let $\rho(x)$ be the expression of x as a word in $\underline{\boldsymbol{m}}$.

Then the following is a presentation for G:

$$\langle \underline{\boldsymbol{h}} \cup \underline{\boldsymbol{m}} \mid R_1 \cup R_3 \cup R_4 \rangle$$

where $R_3 = \{r(\underline{\boldsymbol{h}})/\rho(r(\underline{\boldsymbol{g}})) \mid r \in R_2\}$ and $R_4 = \{m_i^{h_j}/\rho(n_i^{g_j}) \mid i, j\}$.

Proof: It is easily seen that the relations all hold in G. To show that the presentation does not define a larger group, observe that the relations in R_4 ($h_j^{-1} m_i h_j =$ word in $\underline{\boldsymbol{m}}$ implies $m_i h_j = h_j \cdot$ word in $\underline{\boldsymbol{m}}$) permit us to write every element in the presented group as a word in $\underline{\boldsymbol{h}}$ with a word in $\underline{\boldsymbol{m}}$. The relations in R_3 show that (up to changes in $\underline{\boldsymbol{m}}$) every word in $\underline{\boldsymbol{h}}$ can be transformed to one of $|G/N|$ possibilities. The relations in R_1 similarly reduce the words in $\underline{\boldsymbol{m}}$ to $|N|$ classes. Thus the presentation defines a group of order $\leq |G|$. □

Using this lemma and a composition series of G, we can form a presentation for G based on presentations of the composition factors (see the next section for these).

PERFORMANCE III.42: In practice one often gets a nicer presentation and faster performance by using a chief series of G and using the (obvious) presentations for direct products of simple groups.

Pc presentations

It is worth noticing a special case of this which occurs when G is solvable. Then the composition series consists of cyclic factors of prime order and we trivially get a presentation $\langle g | g_p = 1 \rangle$ for these cyclic factors. We therefore get a presentation of the following form:

- Assuming the composition series is $G = G_0 > G_1 > \ldots > G_n = \langle 1 \rangle$, we have generators $g_1, \ldots, g_n$ with $G_{i-1} = \langle G_i, g_i \rangle$.
- We get *power relations* for R_3: If $\left[G_{i-1}{:}G_i\right] = p_i$, we have that $g_i^{p_i}$ can be expressed as a word in the generators g_{i+1} and following
- For R_4 we get *conjugacy relations*: If $i < j$ we have that $g_j^{g_i}$ can be expressed as a word in g_{i+1} and following. (In fact one can do better if the group is supersolvable and even better if it is nilpotent.)

DEFINITION III.43: Such a presentation is called a *PC-presentation* (with "PC" standing alternatively for "polycyclic", "power-conjugate" or "power-commutator".)

We observe that the relations in R_4 permit us to order generators, the power relations in R_3 restrict exponents. Thus every element of G can be written (uniquely) as a product $g_1^{e_1} g_2^{e_2} \cdot \cdots \cdot g_n^{e_n}$ with $0 \leq e_i < p_i$.

We can thus represent group elements by an *exponent vector* $[e_1, \ldots, e_n]$ which is a very compact form of storage. Section IV.3 will describe how one can perform arithmetic operations on such exponent vectors.

In GAP one can convert a (solvable) permutation group into such a form using the command `IsomorphismPcGroup`.

The simple case

While we could easily write down a presentation for a cyclic factor, in general one will still need presentations for the simple composition factors.

One way (which is currently used in GAP) is to use the method of section III.10. For small composition factors this produces reasonable presentations (albeit nothing to boast about).

A much better approach is — mirroring how one would prove theorems — to use the vast amount of theoretical information that has been obtained (for example in the course of the classification of finite simple groups) about simple groups.

If we go through the classes of nonabelian finite simple groups, the following information is found in the literature:

Alternating Group It is a not too hard exercise to show that for odd n, A_n is generated by the elements $g_1 = (1,2,n)$, $g_2 = (1,3,n)$, …, $g_{n-2} = (1,n-1,n)$ and that

$$\langle g_1, \ldots, g_{n-2} \mid \forall i, j > i : g_i^3 = (g_i g_j)^2 = 1 \rangle$$

is a presentation.

Groups of Lie Type This class includes the groups coming from matrix groups, such as $PSL_n(q)$. The unified way to construct these groups also offers a "generic" way to write down a presentation ("Steinberg-presentation").

Sporadic Groups Finally there are 26 so-called "sporadic" groups that do not fit in the previous classes (they include for example the Mathieu groups). For these ad-hoc presentations are known.

An excellent source for such information is the ATLAS of simple groups [CCN+85].

Given a simple composition factor A, we construct an isomorphism (for example by a variant of algorithm III.15) to an isomorphic group B in "nice" form, for which we can just write down the presentation. This lets us transfer the presentation to A.

We will see more efficient ways of constructing such isomorphisms later in section VII.3.

Alas very little of this approach is actually implemented.

Upgrading Permutation group algorithms to Las Vegas

We have seen before in section II.1 that fast algorithms for permutation groups rely on randomized computation of a stabilizer chain and therefore may return a wrong result. To rectify this one would like to have a subsequent step that will verify that the chain is correct. If not, we then can simply continue with further random elements until a renewed test verifies correctly.

Such an algorithm is sometimes called a "Las Vegas" algorithm (in analogy to "Monte Carlo" algorithms): We have a randomized computation of a result that may be wrong, but can do a subsequent verification. (The runtime of such an algorithm thus is good in average, but can be unbounded in the worst case of repeated verification failure.)

The basic approach is the following (again much of the later steps is not implemented):

1. Compute a randomized stabilizer chain for G.
2. Using this chain compute a composition series. (As part of this we get for each factor $G_i > G_{i+1}$ in this series an epimorphism $G_i \to G_i/G_{i+1}$.)
3. Using constructive recognition of the simple factors (see VII.3), write down a presentation for each simple factor F.

4. Use the method of lemma III.41, construct a presentation for G. If the initial chain was too small this is in fact a presentation for a smaller group.
5. Verify that the elements of G actually fulfill the presentation.

To obtain a good runtime complexity for permutation group algorithms in general, we want these steps all to be "fast" (in terms of the degree of the initial permutation group G). This means in particular: We need to be able to construct isomorphisms for the simple factors "quickly" (which in fact has been proven) and need to obtain "short" presentations for the simple factors (basically of relator length $\log^2 |F|$).

The Steinberg presentations mentioned in the last section do not fulfill this, but for almost all cases short variants are known [BGK+97, HS01]. Only the so-called Ree-groups (Lie Type ${}^2G_2$) are missing so far.

Chapter

VII

Group Recognition and Matrix groups

We have seen already one way of working with matrix groups by considering them as permutation groups on a set of vectors. While thius approiach works it can yield an exceedingly large degree and thus is not feasible for larger examples.

Instead, when given a matrix group $G \leq \mathrm{GL}_n(q)$, we want to determine a composition series (or chief series) of G. With such a series in hand the methods of the previous chapter can become feasible.

(I should mention that much of this chapter is still subject of current research and therefore comparatively little is available in implementations.)

VII.1 Towards a Composition Series

The basic idea is to mimic the approach for permutation groups II.6:

> Given a matrix group G, prove that G is simple or find a homomorphism (which we can evaluate) $\varphi\colon G \to H$ such that H is a matrix group of degree not larger than G and that $N := \mathrm{Kern}\,\varphi > \langle 1 \rangle$.

For permutation groups the crucial step towards this was the reduction to primitive groups and the O'Nan-Scott theorem describing the structure of primitive groups. For matrix groups we will use reducibility of the natural module and Aschbacher's theorem (section VII.2).

There is the additional difficulty of how to obtain the kernel N of a homomorphism. For permutation groups we could do this using a stabilizer chain for which we don't have a feasible analogue.

Instead we will first process G/N to the point where we have determined a composition series and from this a presentation for G/N. We then evaluate the relators for G/N in preimages of the generators. This will yield normal subgroup generators for N.

We will then assume that N is generated by "a few" G-conjugates of these normal subgroup generators. To verify correctness we therefore finally need to show that the group generated by these conjugates is normal in G which we can do once we have an element test for N. Such a test again builds on a composition series and decomposition into generators for the simple factors.

As in this situation many of the algorithms we use will use (Monte Carlo) randomization and might return a wrong result. As with permutation groups we therefore use the composition series to determine a presentation and verify relators.

A second feature of this process is that we will be working in a homomorphic image, the homomorphism defined via some action, and will have to take preimages of elements under the homomorphism. As we don't have an easy way of decomposing into generators we therefore keep track of all operations done in the homomorphic image. We therefore know for every element x of this iomage how to express x as word in the generators. To obtain the pre-image of x we then simply evaluate this word in the original groups generators.

Performance VII.1: De facto we will not store words, but "straight line programs" which have smaller storage and faster evaluation. The ide is essentially to store an expression such as $b((ab)^4 * b)^2 * (ab)^2$ not as word $bababababab^2abababab^2abab$ but as "expression tree", storing $c = ab$, $d = (ab)^2 = c^2$, $e = (ab)^4 = d^2$ $f = eb$ and then express the word as bf^2d.

As with permutation groups there has been much interest in complexity aspects of these algorithms: Eventually we want a low-degree polynomial complexity in n and $\log(q)$. One potential difficulty with this is the case of large order cyclic factors. To test membership in such a group we need to solve a discrete logarithm problem, which is a known hard problem.

VII.2 Aschbacher's theorem

In its original form Aschbacher's theorem [Asc84] is a description of maximal subgroups of a matrix group. Since the full matrix group is $\mathrm{GL}_n(q)$ we can — analogous to theorem II.73 — also read it as a statement about matrix groups that are not the full general linear group.

In the following we assume that $G \leq \mathrm{GL}_n(q)$ is a matrix group and $M = \mathbb{F}_q^n$ its natural module.

The theorem defines a series of classes $C_1, \ldots, C_8$ of subgroups, each offering a reduction of the natural module. The (very technical) proof is essentially to show that if a subgroup is not in classes C_1 to C_8 it must be in the class (called C_9) of almost simple groups.

The classes are roughly[1] defined as follows:

C_1 M is reducible.

[1] I'm leaving out various technical conditions

C_2 M is the direct sum of isomorphic subspaces $M = \oplus_{V \in \mathcal{V}} V$ and the action of G permutes these subspaces.

C_4 M is the tensor product of two nonisomorphic submodules.

C_7 M is the tensor product of isomorphic spaces $M = \otimes_{V \in \mathcal{V}} V$ which are permuted under the action of G.

C_3 Up to scalars, we can write G in smaller dimension over a larger field.

C_5 Up to scalars we can do a base change to write G over a smaller field.

C_6 The group G is normalizing a p-group of form $p.p^k$.

C_8 The group G is stabilizing a bilinear or quadratic form.

Note that most of the classes yield an obvious factor group (and suitable action for obtaining an epimorphism onto this factor).

Algorithmically, algorithms exist that with high probability will recognize the various cases (for example the MeatAxe recognizes class C_1). If no class is recognized we assume that G is almost simple and then try to process it as an almost simple group.

VII.3 Constructive Recognition

Assume that we have a group G given by generators of which we believe that it is almost simple. For this group we would like to the following:

Recognition Find out the isomorphism type (with high probability). We can do this in practice (using **heavy** theory) by looking at the distribution of orders for (pseudo-)random elements.

Constructive Recognition Given the (likely) isomorphism type of G construct an effective homomorphism (we have a method to compute images and preimages of elements) from G to a "gold-plated" nice version (typically the "natural" definition) of this group.

Work in the nice version Using the treasure of knowledge built up in theory, we want to find out "everything" (classes, subgroups, presentation, &c.) about the nice group and use the isomorphism to translate the information back to G.

Verify the isomorphism Express known generators of the nice groups as words in images of the generators of G. Express the images of the generators of G as words in the nice generators. Evaluate a presentation for the nice group in G.

While we are describing these processes in the context of matrix groups the transfer of data buit using theory from a "nice" group back to an almost simple group G often is what is needed in lifting algorithms to deal with the radical factor group.

The heart of such functionality is the constructive recognition. Methods for this have been proposed for many classes of simple groups. Some of these assume that G is alreeady given in a particular representation (such as: S_n in the action on pairs of numbers), some do not assume anything about G, but only the fact that we can do element arithmetic and compare elements and have a bound for $|G|$ (a so-called "black-box" group).

The crucial step of such an algorithm then is to recreate the underlying geometry or combinatorics of the group (in the case of (P) GL the vector space) based on properties of group elements.

For example $\mathrm{GL}_n(q)$ is generated by matrices A such that $A - A^0$ has exactly one nonzero entry. Such elements are called "transvections" in the geometric context. We now want to

- Find such elements (and prove that we can find them using random search in reasonable time)
- Identify such elements using only black-box operations.

Doing this requires substantial knowledge about classical groups.

Bibliography

[Asc84] M. Aschbacher, *On the maximal subgroups of the finite classical groups*, Invent. Math. **76** (1984), no. 3, 469–514.

[Atk75] M. Atkinson, *An algorithm for finding the blocks of a permutation group*, Math. Comp. (1975), 911–913.

[BE99] Hans Ulrich Besche and Bettina Eick, *Construction of finite groups*, J. Symbolic Comput. **27** (1999), no. 4, 387–404.

[BGK+97] László Babai, Albert J. Goodman, William M. Kantor, Eugene M. Luks, and Péter P. Pálfy, *Short presentations for finite groups*, J. Algebra **194** (1997), 97–112.

[BLS97] László Babai, Eugene M. Luks, and Ákos Seress, *Fast management of permutation groups. I*, SIAM J. Comput. **26** (1997), no. 5, 1310–1342.

[Boo57] William Boone, *Certain simple, unsolvable problems of group theory. VI*, Nederl. Akad. Wet., Proc., Ser. A **60** (1957), 227–232.

[BP04] László Babai and Igor Pak, *Strong bias of group generators: an obstacle to the "product replacement algorithm"*, J. Algorithms **50** (2004), no. 2, 215–231, SODA 2000 special issue.

[Cam99] Peter J. Cameron, *Permutation groups*, London Mathematical Society Student Texts, vol. 45, Cambridge University Press, 1999.

[Can73] John J. Cannon, *Construction of defining relators for finite groups*, Discrete Math. **5** (1973), 105–129.

[CCH01] John Cannon, Bruce Cox, and Derek Holt, *Computing the subgroup lattice of a permutation group*, J. Symbolic Comput. **31** (2001), no. 1/2, 149–161.

[CCN+85] J[ohn] H. Conway, R[obert] T. Curtis, S[imon] P. Norton, R[ichard] A. Parker, and R[obert] A. Wilson, *ATLAS of finite groups*, Oxford University Press, 1985.

[CELG04] John J. Cannon, Bettina Eick, and Charles R. Leedham-Green, *Special polycyclic generating sequences for finite soluble groups*, J. Symbolic Comput. **38** (2004), no. 5, 1445–1460.

[CH04] John Cannon and Derek Holt, *Computing maximal subgroups of finite groups*, J. Symbolic Comput. **37** (2004), no. 5, 589–609.

[CLGM+95] Frank Celler, Charles R. Leedham-Green, Scott H. Murray, Alice C. Niemeyer, and E. A. O'Brien, *Generating random elements of a finite group*, Comm. Algebra **23** (1995), no. 13, 4931–4948.

[CNW90] Frank Celler, Joachim Neubüser, and Charles R. B. Wright, *Some remarks on the computation of complements and normalizers in soluble groups*, Acta Appl. Math. **21** (1990), 57–76.

[Deh11] Max Dehn, *Über unendliche diskontinuierliche Gruppen*, Math. Ann. **71** (1911), 116–144.

[Dix69] John D. Dixon, *The probability of generating the symmetric group*, Math. Z. **110** (1969), 199–205.

[DM88] John D. Dixon and Brian Mortimer, *The primitive permutation groups of degree less than 1000*, Math. Proc. Cambridge Philos. Soc. **103** (1988), 213–238.

[DM96] _____, *Permutation groups*, Graduate Texts in Mathematics, vol. 163, Springer, 1996.

[EH01] Bettina Eick and Alexander Hulpke, *Computing the maximal subgroups of a permutation group I*, Proceedings of the International Conference at The Ohio State University, June 15–19, 1999 (Berlin) (William M. Kantor and Ákos Seress, eds.), Ohio State University Mathematical Research Institute Publications, vol. 8, de Gruyter, 2001, pp. 155–168.

[GP06] Alexander Gamburd and Igor Pak, *Expansion of product replacement graphs*, Combinatorica **26** (2006), no. 4, 411–429.

[GS90] Stephen P. Glasby and Michael C. Slattery, *Computing intersections and normalizers in soluble groups*, J. Symbolic Comput. **9** (1990), 637–651.

[Hal33] Philip Hall, *A contribution to the theory of groups of prime-power orders*, Proc. Lond. Math. Soc. II. Ser **36** (1933), 29–95.

[HEO05] Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien, *Handbook of Computational Group Theory*, Discrete Mathematics and its Applications, Chapman & Hall/CRC, Boca Raton, FL, 2005.

[HP89] Derek F. Holt and W. Plesken, *Perfect groups*, Oxford University Press, 1989.

[HS01] Alexander Hulpke and Ákos Seress, *Short presentations for three-dimensional unitary groups*, J. Algebra **245** (2001), 719–729.

[Hul98] Alexander Hulpke, *Computing normal subgroups*, Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation (Oliver Gloor, ed.), The Association for Computing Machinery, ACM Press, 1998, pp. 194–198.

[Hul99] _____, *Computing subgroups invariant under a set of automorphisms*, J. Symbolic Comput. **27** (1999), no. 4, 415–427, (ID jsco.1998.0260).

[Hul00] _____, *Conjugacy classes in finite permutation groups via homomorphic images*, Math. Comp. **69** (2000), no. 232, 1633–1651.

[Hul05] _____, *Constructing transitive permutation groups*, J. Symbolic Comput. **39** (2005), no. 1, 1–30.

[Kan85] William M. Kantor, *Sylow's theorem in polynomial time*, J. Comput. System Sci. **30** (1985), no. 3, 359–394.

[KC07] D. Kunkle and G. Cooperman, *Twenty-six moves suffice for Rubikfbs Cube.*, Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC fb07), ACM Press, 2007.

[Leo80] Jeffrey S. Leon, *On an algorithm for finding a base and a strong generating set for a group given by generating permutations*, Math. Comp. **35** (1980), no. 151, 941–974.

[LM02] Eugene M. Luks and Takunari Miyazaki, *Polynomial-time normalizers for permutation groups with restricted composition factors*, Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (Teo Mora, ed.), The Association for Computing Machinery, ACM Press, 2002, pp. 176–183.

[LPS87] Martin W. Liebeck, Cheryl E. Praeger, and Jan Saxl, *A classification of the maximal subgroups of the finite alternating and symmetric groups*, J. Algebra **111** (1987), 365–383.

[LPS88] _____, *On the O'Nan-Scott theorem for finite primitive permutation groups*, J. Austral. Math. Soc. Ser. A **44** (1988), 389–396.

[LS95] Martin W. Liebeck and Aner Shalev, *The probability of generating a finite simple group*, Geom. Dedicata **56** (1995), no. 1, 103–113.

[Luk82] Eugene M. Luks, *Isomorphism of graphs of bounded valence can be tested in polynomial time*, J. Comput. System Sci. **25** (1982), no. 1, 42–65.

[Min98] Torsten Minkwitz, *An algorithm for solving the factorization problem in permutation groups*, J. Symbolic Comput. **26** (1998), no. 1, 89–95.

[MO95] Scott H. Murray and E. A. O'Brien, *Selecting base points for the Schreier-Sims algorithm for matrix groups*, J. Symbolic Comput. **19** (1995), no. 6, 577–584.

[MSWZ93] Jean-François Mestre, René Schoof, Lawrence Washington, and Don Zagier, *Quotients homophones des groupes libres*, Experiment. Math. **2** (1993), no. 3, 153–155.

[Neu60] Joachim Neubüser, *Untersuchungen des Untergruppenverbandes endlicher Gruppen auf einer programmgesteuerten elektronischen Dualmaschine*, Numer. Math. **2** (1960), 280–292.

[Neu86] Peter M. Neumann, *Some algorithms for computing with finite permutation groups*, Groups – St Andrews 1985 (Edmund F. Robertson and Colin M. Campbell, eds.), Cambridge University Press, 1986, pp. 59–92.

[Neu87] _____, *Berechnungen in Gruppen*, Vorlesungsskript ETH Zürich, 1987.

[Nie94] Alice C. Niemeyer, *A finite soluble quotient algorithm*, J. Symbolic Comput. **18** (1994), no. 6, 541–561.

[Nov55] P. S. Novikov, *Ob algoritmičeskoĭ nerazrešimosti problemy tožděstva slov v teorii grupp [on the algorithmic unsolvability of the word problem in group theory]*, Trudy Mat. Inst. im. Steklov. no. 44, Izdat. Akad. Nauk SSSR, Moscow, 1955.

[O'B90] E[amonn] A. O'Brien, *The p-group generation algorithm*, J. Symbolic Comput. **9** (1990), 677–698.

[Pas68] Donald Passman, *Permutation groups*, W. A. Benjamin, Inc., New York-Amsterdam, 1968.

[Ple87] W. Plesken, *Towards a soluble quotient algorithm*, J. Symbolic Comput. **4** (1987), no. 1, 111–122.

[RDU03] Colva M. Roney-Dougal and William R. Unger, *The affine primitive permutation groups of degree less than 1000*, J. Symbolic Comput. **35** (2003), 421–439.

[Rem30] Robert Remak, *Über die Darstellung der endlichen Gruppen als Untergruppen direkter Produkte*, J. Reine Angew. Math. **163** (1930), 1–44.

[Rok08] Thomas Rokicki, *Twenty-five moves suffice for Rubik's Cube*, arXiv:0803.3435v1, 2008.

[Ser03] Ákos Seress, *Permutation group algorithms*, Cambridge University Press, 2003.

[Sim70] Charles C. Sims, *Computational methods in the study of permutation groups*, Computational Problems in Abstract Algebra (John Leech, ed.), Pergamon press, 1970, pp. 169–183.

[Sim90] _____, *Computing the order of a solvable permutation group*, J. Symbolic Comput. **9** (1990), 699–705.

[Sim94] Charles C. Sims, *Computation with finitely presented groups*, Cambridge University Press, 1994.

[The97] Heiko Theißen, *Eine Methode zur Normalisatorberechnung in Permutationsgruppen mit Anwendungen in der Konstruktion primitiver Gruppen*, Dissertation, Rheinisch-Westfälische Technische Hochschule, Aachen, Germany, 1997.

[Wam74] J. W. Wamsley, *Computation in nilpotent groups (theory)*, Proceedings of the Second International Conference on the Theory of Groups (M. F. Newman, ed.), Lecture Notes in Mathematics, vol. 372, Springer, 1974, pp. 691–700.