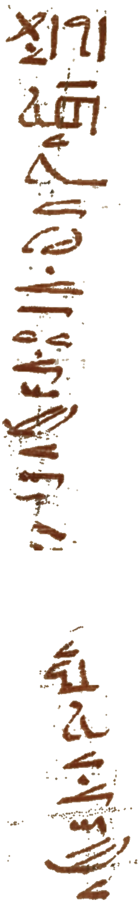
# Notes on Computational Group Theory

Alexander Hulpke

Spring 2024



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

*Title graphics:*

Correct method of reckoning,
for grasping the meaning of things
and knowing everything that is,
obscurities ... and all secrets.

Rhind Mathematical Papyrus
THE BRITISH MUSEUM

These notes are accompanying my course M676, Computational Group Theory, held Spring 2024 at Colorado State University.

# Contents

# Preface

This are lecture notes I prepared for a course on Computational Group Theory which I taught in the Springs of 2006, 2010, and 2024 at Colorado State University. The audience consisted mostly of graduate students in their second year and later, thus the course could assume a first year algebra course as prerequisite. Part of the notes also was used in an REU at the University of Arizona in Summer 2008.

My aim in this course was to give an overview over *most* of computational group theory from the point of view of understanding the principle behind calculations and understand what kinds of calculations are easy, hard or infeasible.

In many cases the presentation, the prominence given to particular algorithms or classes of groups, or depth of description is hopelessly biased by my personal preferences and the desire to use this course to prepare students for dissertation work with me.

In particular, as only few of the students had a background in computer science, I decided to essentially eliminate all references to complexity and in a few cases (which I hope all carry an explicit warning about this) even replace polynomial time algorithms with potentially exponential ones as long as the run time in practice is not too bad.

Another main divergence from “classical” descriptions is the lack of a chapter on polycyclic presentations. Instead these are treated with their arithmetic as a special case of rewriting systems, in their algorithms in the more general area of “lifting” algorithms using homomorphic images.

These notes are deeply indebted to Derek Holt’s marvellous “Handbook of Computational Group Theory” [HEO05], which is often quite detailled – a terrific fact if one wants to implement the algorithms — necessitating more time for explanation than a one-semester course can allocate.

Besides Holt’s book I have freely borrowed from and am indebted to Ákos Serres’ work on permutation groups [Ser03], Charles Sims’ tome on finitely presented groups [Sim94], lecture notes by Peter Neumann [Neu87] and notes I took in lectures of my advisor, Joachim Neubüser.

I apologize in advance if these references are not always explicitly listed, but after all these are lecture notes. Similarly I have not aimed to make the bibliography exhaustive. There are a few references to the research literature but I apologize for any ommissions.

I would like to acknowledge feedback and corrections from many colleagues and students, in particular Thomas Breuer, Klaus Lux, Kenneth Monks, Soley Jonsdottir, and Ellen Ziliak.

Some work on these lecture notes has been done with support from the National Science Foundation under Grant No. 0633333, which is gratefully acknowledged. Any opinions, findings and conclusions or recomendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

You are welcome to use these notes freely for your own courses or students – I'd be indebted to hear if you found them useful.

Fort Collins, Spring 2024
Alexander Hulpke
`hulpke@math.colostate.edu`

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 (and one of the millennium problems) 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, E. 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 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 64-bit system; other implementations will 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 8 bytes on a 64 bit system.)

**Small Numbers:** All processors have built in arithmetic for small numbers (up to 64 bits). We will use this arithmetic for such small numbers. (In fact for technical reasons the limit in GAP is $\pm 2^{60}$. There is a notable slowdown if numbers get above $2^{60}$.)

**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 $n \le 2^{16}$ 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 in the computer 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. In 2GB we thus could store about 1 million permutations. On the other hand if we have permutation degree 100000, we could only store 500 permutations in the same amount of memory.

As we want to be able to work with groups that are even larger (in order and storage size), we clearly are only able to store a small proportion of group elements.

## I.3 Orbits and Stabilizers

### Left and Right

**lichtung**
manche meinen
lechts und rinks
kann man nicht velwechsern.
werch ein illtum!

Ernst Jandl

**dilection**
some think
terring reft flom light
is a piece of cake
boy are they evel
long!

Translation: Anselm Hollo

In these notes we will always have groups acting on the right, and consider right cosets and row vectors. Consequenctially the product of permutations is $(1,2)\cdot(2,3)=(1,3,2)$. 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.

### 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 on a set $\Omega$ 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 $\omega^G$ 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 $\left|\omega^G\right| = \left[G\colon \mathrm{Stab}_G(\omega)\right]$.

If $G$ acts on $\Omega$, we get an homomorphism $\varphi\colon G \to S_{|\Omega|}$, 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 $g \in G$.

## 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 $\omega^G$. On the other hand property b) shows that $\Delta$ 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 $\Omega$. Also a point $\omega \in \Omega$.

**Output:** return the orbit $\omega^G$.

```
begin
 1: $\Delta := [\omega]$;
 2: for $\delta \in \Delta$ do
 3:    for $i \in \{1, \ldots, m\}$ do
 4:       $\gamma := \delta^{g_i}$;
 5:       if $\gamma \notin \Delta$ then
 6:          Append $\gamma$ to $\Delta$;
 7:       fi;
 8:    od;
 9: od;
10: return $\Delta$;
end
```

Note that the **for**-loop in line 2 runs also through elements added to teh partial orbit $\Delta$ in the course of the algorithm.

Note I.5: Instead of starting with $\omega$ 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.)

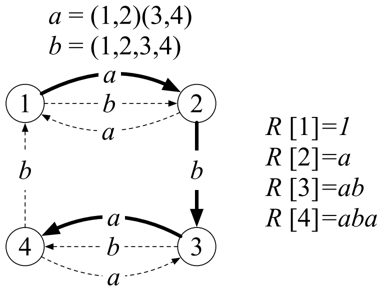


Figure I.1: The orbit of 1 under $D_8 = \langle a = (1,2)(3,4), b = (1,2,3,4)\rangle$

## Representatives

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

Such a list of representatives is called a *transversal*. By lemma I.1 it simultaneously is a set of representatives for the cosets of $\mathrm{Stab}_G(\omega)$. We shall calculate it in parallel to the orbit computation.

As for notation, it makes sense to consider the orbit $\omega^G$ as a list (with fixed ordering) to maintain a correspondence between orbit points and their representatives.

To simplify notation, we will simply index the transversal with orbit elements: $R[\delta]$. By this we mean $R[i]$ 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 $\delta$ to $\gamma$ 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 $\delta$, then $xg$ is a representative for $\delta^g$. This gives the following extension of the orbit algorithm:

Algorithm I.9: Orbit algorithm with transversal computation

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

**Output:** return the orbit $\omega^G$ and a transversal $R$.

```
begin
1: $\Delta := [\omega]$;
2: $R := [1]$;
3: for $\delta \in \Delta$ do
4:     for $i \in \{1, \dots, n\}$ do
```

```
5:        $\gamma := \delta^{g_i}$;
6:        **if** $\gamma \notin \Delta$ **then**
7:            Append $\gamma$ to $\Delta$;
8:            Append $R[\delta] \cdot g_i$ to $R$;
9:        **fi**;
10:    **od**;
11: **od**;
12: **return** $\Delta$, $R$;
**end**
```

This is illustrated in Figure I.1, in which we calculate the orbit of the point 1 under the group $D_8$, generated by the two permutations $a = (1,2)(3,4)$ and $b = (1,2,3,4)$. The bold lines indicate the first time an image is obtained, defining the transversal elements.

NOTE I.10: It is worth observing that the representative $R[\delta]$ 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 show the maximum number of moves for RUBIK's cube is 20, see [KC07, Rok10] and `cube20.org`.

## 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 $\gamma$ precedes $\delta$ 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 $\delta$ is obtained as image $\delta = \gamma^g$ of a known point $\gamma$.

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: For efficiency, it is desirable that the number of products to be formed 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 extension 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 $G = \left\langle \underline{g} \right\rangle$ a finitely generated group and $S \leq G$ with $\left[G{:}S\right] < \infty$. Suppose that $\underline{r} = \{r_1, \dots, r_n\}$ 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 $x \in G = \left\langle \underline{g} \right\rangle$ 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. This term means that we take the given expression for $x$ as a symbol string in the generators, and change it iteratively according to a certain scheme. 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 $\omega$ as coset representatives. (The representative for the coset $Sg$ is $T[\omega^g]$.)

We thus get the following algorithm:

ALGORITHM I.17: Orbit/Stabilizer algorithm

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

```
Output: return the orbit $\omega^G$, 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 $\gamma$ to $\Delta$;
9:              Append $T[\delta] \cdot g_i$ to $T$;
10:         else
11:             $S := \left\langle S, T[\delta] \cdot g_i \cdot T[\gamma]^{-1} \right\rangle$;
12:         fi;
13:     od;
14: od;
15: return $\Delta$, $T$, $S$;
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).

Note I.19: if the orbit contains $\left|\omega^G\right| = \left[G{:}S\right]$ many points, the algorithm is forming $\left|\omega^G\right| \cdot \left|\underline{\mathbf{g}}\right|$ many images (the image of every point under every generator), of those $\left|\omega^G\right| - 1$ are new. Thus there are

$$\left|\omega^G\right| \cdot \left|\underline{\mathbf{g}}\right| - \left(\left|\omega^G\right| - 1\right) \left|\omega^G\right| \cdot \left|\underline{\mathbf{g}}\right| - \left|\omega^G\right| + 1 = \left|\omega^G\right| \cdot \left(\left|\underline{\mathbf{g}}\right| - 1\right) + 1 = \left[G{:}S\right] \cdot \left(\left|\underline{\mathbf{g}}\right| - 1\right) + 1$$

Schreier generators.

Performance I.20: We will see later (note III.33 in chapter III) that the rather large number of Schreier generators $\left[G{:}S\right] \cdot \left(\left|\underline{\mathbf{g}}\right| - 1\right) + 1$ 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[1] 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.

  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.

If we know the normal structure of a group improvements are possible, see section II.4.

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

[1]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].

**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 $\langle \underline{\boldsymbol{n}} \rangle = \langle U \rangle_G$. □

### Consequence: What to ask for?

The ability to calculate orbits, essentially at a cost proportional to the length of the orbit, influences the design of other algorithms. If there is a natural group action defined, it is sufficient to compute and return only a list of representatives, from these one could obtain all objects as orbits of the representatives.

Doing so not only makes the output size smaller, but typically also saves substantial memory and computing time. In general algorithms of this type do not only determine representatives, but also their stabilizers (of course not computed by an orbit algorithm), knowing them for example use information about the orbit length

Typical examples of this are group elements – conjugation by the group forms orbits, called conjugacy classes. Instead of enumerating all elements, it is sufficient to list only representatives of the classes. The stabilizer of an element then is the centralizer.

When dealing with subgroups of a group similarly conjugacy forms orbits. A typical computation will determine subgroups only up to conjugacy, the stabilizers here are the normalizers. Some prominent classes of subgroups, such as Sylow subgroups also typically are computed via single representatives.

## I.4 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. (GAP for example uses the *Mersenne Twister* algorithm, `http://en.wikipedia.org/wiki/Mersenne_twister`.) 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.22: (Pseudo)Random, "Product Replacement"

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

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 $r = \max(11, |\underline{g}|)$ group elements and one extra group element $a$. (Experiments show that one needs $r \geq 10$.)

```
PSEUDORANDOM()
begin
 1: $s := \text{RANDOM}([1..r])$;{pick two random list elements}
 2: $t := \text{RANDOM}([1..r] \setminus [s])$;
 3: $e := \text{RANDOM}([-1,1])$;{random choice of product/quotient}
 4: if $\text{RANDOM}([1,2]) = 1$ 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}
  $k := |\underline{g}|$;
  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 8/9 of the PSEUDORANDOM routine ensures a quick growth of word lengths.

## I.5 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(`$\omega$`,`$g$`)`, which will return the image $\omega^g$ for the particular definition of the action. No[2] 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

```
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);
```

[2] well, almost no

`RepresentativeAction(`$G$`,`$\omega$`,`$\delta$`,actfun);`,

`Stabilizer(`$G$`,`$\omega$`,actfun);`,

`ActionHomomorphism(`$G$`,`$\Omega$`,actfun,"surjective");`] returns a homomorphism from $G$ to the permutation action on $\Omega$ (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 $\Omega$ in which the orbit lies: $\Omega$ might be small and afford a cheap indexing function – in this case the position in $\Omega$ can be used for lookup. Alternatively, $\Omega$ 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[3].
All action functions therefore take (a superset of) the domain as an optional second argument, e.g. `Orbit(`$G$`,`$\Omega$`,`$\omega$`,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 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(`$G$`,`$\omega$`,`$\underline{\boldsymbol{g}}$`,`$\underline{\boldsymbol{h}}$`,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 $G$.

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

## 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[4] 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.

[4] 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

Thirsty to see what God would see,
Judah Loew gave in to permutations
with letters in such complex variations
that he at last uttered the Name that is Key.

Portal, Echo, Host and Palace

Translation: Matias Giovannini

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 $\beta_i$ 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 $\beta_{i+1}$ under $G^{(i)}$ (we shall use the convention that $\beta_{i+1} =$ `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: $\beta_i$ 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 need a base to contain particular points, which we would chose first.

A naive way to calculate a stabilizer chain would be to simply compute the orbit of $\beta_1$ 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. Overall this would result in a run time that is exponential in the number of points. The way around this problem is to use the partially constructed stabilizer chain to remove redundant elements. We therefore consider element tests first.

## Element Test

The basic idea towards an element test is the following algorithm which, given a stabilizer chain and a group element $x$, writes $x$ as a 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 $L = [b_1, b_2, \dots, b_m]$ 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 $\beta$. Thus at the end of the algorithm, after dividing off representatives, we must have $g = 1$. □

A small modifiication of this algorithm now lets us do an element test for the group represented by the chain. 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 $\delta$ may not be in the orbit of $\beta$. (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**
  $L := []$;
  **while** $C$.generators <> [] **do**
      $\beta := C$.orbit[1];
      $\delta = \beta^g$;
      **if** $C$.transversal[$\delta$] does not exist **then**
          **return** not contained;
      **fi**;
      $r := C$.transversal[$\delta$];
      $g := g/r$;
      Add $r$ to $L$;
      $C := C$.stabilizer;
  **od**;
  **if** $g \neq ()$ **then**
      **return** not contained;
  **else**
      **return** $L$
  **fi**;
**end**
```

## 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. We also assume that on the current level the `.orbit` and `.transversal` components correspond.

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 $\beta = C.\texttt{orbit}[1]$ the orbit of $\beta$ under $\langle C.\texttt{generators}\rangle$ is $C.\texttt{orbit}$ and that

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

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

Whenever we modify a layer, we will have to ensure that it is a a proper chain, before returning back.

To prove correctness of a stabilizer chain computation the following observations will be useful, it gives a testable condition which ensures correctness of the stabilizer chain.

Lemma II.7: Let $C$ be a layer of a partial stabilizer chain with orbit starting at $\beta = C.\texttt{orbit}[1]$ 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. $C.\texttt{stabilizer}$ is a proper stabilizer chain for $\mathrm{Stab}_G(\beta)$.
2. $|G| = |C.\texttt{orbit}| \cdot |\langle C.\texttt{stabilizer}\rangle|$

Returning to the question of calculating stabilizer chains, 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

[1]The conditions are trivially all necessary.

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.

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 $\underline{g}$ for a permutation group $G$
Output: A recursive data structure for a stabilizer chain for $G$.
begin
1: $C$ := rec(generators := []);
2: for $a \in \underline{g}$ 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:       $\beta$ := 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 $\delta$ 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 $\gamma$ 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 $\gamma \notin O$ then
29:                  Add $\gamma$ to $O$; update transversal;
30:               else
31:                  $s := T[\delta]bT[\gamma]^{-1}$;
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 $U \leq G$ 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 $\omega$ 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 $\bar{g}$ be the product of transversal elements divided off at this point. Then $r = g/\bar{g} \in G^{(i+1)}$. 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.22) 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.

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.11

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!)

[2]a rhetorical "if" as a mathematician

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 $\Gamma = (\gamma_1, \dots, \gamma_n)$ 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 $\Gamma$, preferably at the same position, but even different positions are fine. Call this new base $E$.

Then we add the remaining points of $\Gamma$ 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 $\eta_i$ and $\eta_j$ 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 $\Gamma$ (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 $[\gamma_1, \dots, \gamma_m]$ 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. Recently the tight upper bound of 20 has been proven [Rok10], `http://cube20.org`, 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 $\varphi\colon G \to H$ given by a generating set $\underline{g}$ of $G$ and the images $\underline{g}^{\varphi}$.

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

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 $\{(g, g^{\varphi}) \mid g \in \underline{g}\}$.
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 $\Omega$ then decomposes in $\Omega_G \cup \Omega_H$.

Let $D = \left\langle (g, g^{\varphi}) \mid g \in \underline{g} \right\rangle$ the group (the "diagonal" subgroup of the direct product) representing the homomorphism $\varphi$. Then the point wise stabilizer $\mathrm{Stab}_D(\Omega_H)$

[3]Calculations were done using the spare cycles donated by a Hollywood studio and by Google!

corresponds to the set of elements whose image is trivial, i,e, its $G$-projection is the kernel of $\varphi$.

## 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
- 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 $\beta_{k+1}$ 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 $G = A_4$.

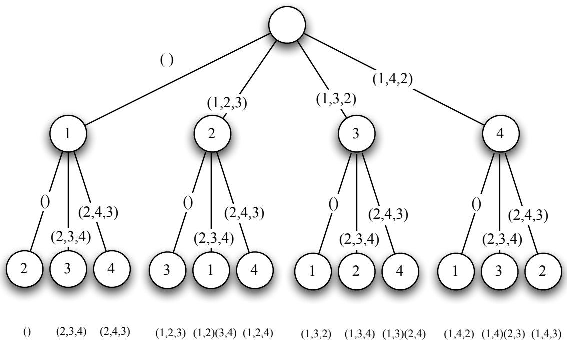
Again, as we consider stabilizer chains as recursive objects, this is a recursive algorithm.

**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: $leaf := |C.\texttt{stabilizer.generators}| = 0$; {have we reached a leaf of the tree?}
2: $\Delta := C.\texttt{orbit}$;
3: for $\delta \in \Delta$ do
4:     $x := C.\texttt{transversal}[\delta]$;
5:     if $leaf$ then
```

[4]and better methods might impact the question of whether P=NP

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



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 $A_4$

```
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 $\Delta$ 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 the group $G = \langle (1,3,5,7)(2,4,6,8), (1,3,8)(4,5,7)\rangle$. 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 | () | √ |
| 2 | 4 | (1,2,4)(5,6,8) | √ |
| 4 | 1 | (1,4,2)(5,8,6) | √ |
| 5 | 6 | (1,5)(2,6)(3,7)(4,8) | √ |
| 6 | 8 | (1,6,4,5,2,8)(3,7) | √ |
| 8 | 5 | (1,8,2,5,4,6)(3,7) | √ |

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 $\Delta \subset \Omega$. (Without loss of generality, assume that $|\Delta| \leq \frac{|\Omega|}{2}$, otherwise we consider the complement $\Omega - \Delta$.) We choose a base whose initial points are chosen from within $\Delta$ as far as possible, say $\beta_1, \ldots, \beta_k \in \Delta$ and $G^{(k)}$ moves no point in $\Delta$. Then clearly $G^{(k)} \leq \mathrm{Stab}_G(\Delta)$. Furthermore the possible images for $\beta_i$ $(i \leq k)$ are restricted to $\Delta$.

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 $\beta_1$ within a long cycle of $x$ whose length $l$ occurs rarely (so there are few cycles in $y$ of this length). Then $\beta_1$ 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 $\beta_1^g$.

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

Note II.20: Newer version of backtrack algorithms, so called "Partition backtrack" routines label the tree not with base images, but with ordered[6] partitions of $\Omega$. The partial base image $(\gamma_1, \ldots, \gamma_k)$ then corresponds to a partition with each $\gamma_i$ $(i \leq k)$ 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.

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

- 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 $\Gamma$ and $\Theta$. 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 $P \le G$.

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 $K \le P$ (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.

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, testing for minimality in left cosets and right cosets. While this does not guarantee minimality in the double coset, it is a reasonable tradeoff between cost and gain.

The first condition uses 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 $\gamma_l$ is minimal in the orbit $\gamma_l^{\mathrm{Stab}_K(\gamma_1,\ldots,\gamma_{l-1})}$.

<u>Proof:</u> 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 uses minimality in the right coset $Kg$.

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 $\gamma_l$ cannot be among the last $s-1$ elements of its orbit under $R$.

<u>Proof:</u> 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 $\gamma_l^R$ must contain at least $s-1$ elements larger than $\gamma_l$. □

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.11) 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 $\varphi\colon G \to S_m$ with $N = \operatorname{Kern}\varphi$.

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* $\varphi$ on $\Omega$, the corresponding statements remain true if we interpret them for the factor $G/\operatorname{Kern}\varphi$.

## 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 $\Gamma$ and $\Delta$ are orbits[7]. 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 of $G^\epsilon$, that

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

[7] or unions of orbits. We do not need that the action on $\Gamma$ and $\Delta$ is transitive.

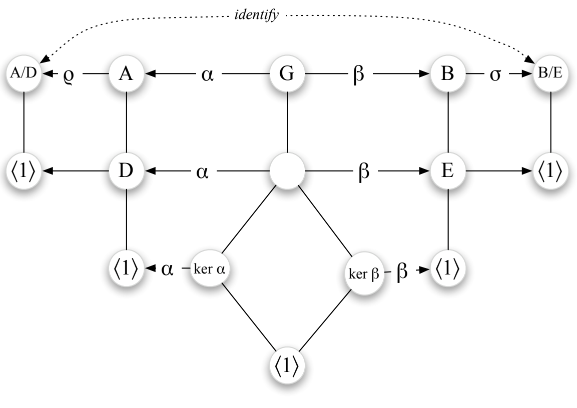


Figure II.2: Subdirect Product

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

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

is called the subdirect product of $A$ and $B$. It is an easy exercise to see that $A \curlywedge B$ 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 $A \curlywedge B$ is misleading: the product also depends on the choice of factor groups as well as on $\zeta$. 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 $A \curlywedge B$ 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 $\zeta$, 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[8]. We consider the subdirect product $A \curlywedge B$ with respect to these homomor-

[8]One can drop the condition that $\rho$ and $\sigma$ have to be surjective by considering subgroups of $A$ and

phisms. Let $G$ be a group which makes the diagram
$$\begin{tikzcd} G \arrow[r, "\alpha"] \arrow[d, "\beta"] & A \arrow[d, "\rho"] \\ B \arrow[r, "\sigma"] & Q \end{tikzcd}$$
commutative (I.e.: There exist homomorphisms $\alpha: G \to A$ and $\beta: G \to B$ such that for every $g \in G$ we have that $g^{\alpha\rho} = g^{\beta\sigma}$). Then there exists a unique map $\mu: G \to A \curlywedge B$, such that the diagram

$$\begin{tikzcd} G \arrow[dr, "\mu"] \arrow[drr, "\alpha"] \arrow[ddr, "\beta"'] & & \\ & A \curlywedge B \arrow[r, "\gamma"] \arrow[d, "\delta"] & A \arrow[d, "\rho"] \\ & B \arrow[r, "\sigma"] & Q \end{tikzcd}$$

(with $\gamma, \delta$ 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 $\mu$ to be the corestriction[9] of $\epsilon$ 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 $\mu$ 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.)

For example, if $A = B = S_3$, there are three possible factor groups – $\langle 1\rangle$, $C_2$ and $S_3$. Setting $Q = \langle 1\rangle$ yields the direct product $\langle (1,2),(4,5),(1,2,3),(4,5,6)\rangle$, $Q = S_3$ yields diagonal subgroups $\langle (1,2)(4,5),(1,2,3)(4,5,6)\rangle$ (or any relabelling of the points). Finally the factor group $Q = C_2$ yields the proper subdirect product $\langle (1,2,3),(4,5,6),(1,2)(4,5)\rangle$ of order 18.

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 $\leq n$).

$B$ instead.

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

## 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 $\Omega$.

DEFINITION II.30: A partition $\mathcal{B} = \{B_1, \ldots, B_k\}$ of $\Omega$ (I.e. we have that $B_i \subset \Omega$ and $\Omega$ 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 $\Omega$ if $G$ acts transitively, and affords a nontrivial block system. Otherwise we say the group acts *primitively*.

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 $B_i^g \cap B_j \neq \emptyset$. 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 $\Omega$ 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 $\Omega$.

Using the convention that $B_1$ is the block containing $\omega$, 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 $B^g \cap B^h \neq \emptyset$. 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 $\Omega$. 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 $\omega$. Then any $g \in G$ which fixes $\omega$ has to fix $B$, thus $\mathrm{Stab}_G(\omega) \leq \mathrm{Stab}_G(B)$.

To show that the map from subgroups to blocks is surjective, let $\omega \in B$ be a block and $\delta \in B$. There is $g \in G$ such that $\omega^g = \delta$. But then $\delta \in B^g$, 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 $\alpha$ are contained in the same block. By running through all possible $\alpha$, 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 $\Omega$ 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 $\alpha^g, \beta^g$ 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 $\omega$. We call $r[\omega]$ the *label* of $\omega$. Then joining two cells is done by simply replacing the label for elements in the second cell by labels for the first cell:

Union($\alpha, \beta$)

```
Input: Two cells, given by their representatives $\alpha$ and $\beta$.
Output: The two cell are joined, representing them by the label for the first cell.
begin
 1: for $\omega \in \Omega$ do
 2:    if $r[\omega] = \beta$ then
 3:       $r[\omega] := \alpha$;
 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 $\Omega$. 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: $\mu := seed[1]$
 4: for $\omega \in \Omega$ do
 5:    if $\omega \in seed$ then
 6:       $r[\omega] := \mu$;
 7:       Add $\omega$ to $q$;
 8:    else
 9:       $r[\omega] := \omega$;
10:    fi;
11: od;
12: $l := 1$;
13: while $l \le |q|$ do
14:    $\gamma := q[l]$; $\delta := r[\gamma]$; {point and its representative}
15:    for $g \in \underline{g}$ do
16:       $\alpha := r[\gamma^g]$;
17:       $\beta := r[\delta^g]$;
18:       if $\alpha \neq \beta$ then {Two points are in the same block but their images are
          not}
19:          UNION$(\alpha, \beta)$; {join block given by $\beta$ to block given by $\alpha$}
20:          Add $\beta$ to $q$; {As $\beta$ block got deleted}
```

```
21:         fi;
22:     od;
23:     $l := l + 1$;
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 $\omega^g$ and $\delta^g$ are in the same cell.

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

> (*) If $\beta$ is the label for a cell, and $\omega$ is in this cell, then $\beta^g$ and $\omega^g$ 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 $\beta$ 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 $\beta$ to the queue, we have to show that doing so is sufficient: Consider a point $\omega$ that is labeled by $\beta$ and suppose we relabel $\omega$ to $\alpha$. This can only happen if we also relabel $\beta$ to $\alpha$ and in this case we enforce (*) for $\beta$ and $\alpha$.

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

This argument also shows that a point $\omega$ 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 $\alpha$ 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 $\Omega$. 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 $\alpha$ which are representatives for the orbits of $\mathrm{Stab}_G(1)$ on $\Omega$, 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.

## Orbits of normal subgroups

A further connection between group structure and block structures is given by the following lemma:

LEMMA II.45: Let $G$ be a transitive group on $\Omega$ and $N \lhd G$. Then the orbits of $N$ form a block system of $G$

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

This gives us a first computational application of the concept of blocks, namely an improvement of the basic Orbit/Stabilizer algorithm in the presence of a normal subgroup $N \lhd G$: Suppose that $G$ acts on $\Omega$ and for $\omega \in \Omega$ we have already computed the orbit $\Delta = \omega^N$ and the stabilizer $S = \mathrm{Stab}_N(\omega)$. Our goal is to determine $\omega^G$ and $\mathrm{Stab}_G(\omega)$.

As $N \lhd G$, lemma II.45 states that $\Delta$ is a block for $G$. We can thus compute the orbit of $\Delta$ (acting on sets via the action on elements) under $G$ in an ordinary Orbit/Stabilizer calculation. The $G$-orbit $\omega^G$ then is simply the union of the sets in the orbit $\Delta^G$. The gain in this approach is that instead calculating images of all points under all generators of $G$, we always first map just a single point of $\Delta$ (or its images) and determine – based on whether the image of this point is already known – whether $\Delta^x$ is new. The number of redundant images (and of Schreier generators) thus is reduced roughly by a factor of $|\Delta|$.

If we want transversal elements for $\omega^G$, we obtain these simply by multiplying teh elements of $N$ with transversal elements for $\Delta^G$.

The stabilizer computed in this algorithm is the set stabilizer $\mathrm{Stab}_G(\Delta)$. Since $N$ fixes and is transitive on $\Delta$, and since $\mathrm{Stab}_N(\omega) = \mathrm{Stab}_G(\omega \cap N)$, we get that $\mathrm{Stab}_G(\Delta)/N \cong \mathrm{Stab}_G(\omega)/\mathrm{Stab}_N(\omega)$, figure II.3.

This means that we can obtain generators for $\mathrm{Stab}_G(\omega)$ by *correcting* generators for $\mathrm{Stab}_G(\Delta)$ with elements from $N$. This correction is simply by an element that will map $\omega^g \in \Delta$ back to $\omega$.

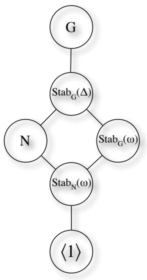


Figure II.3: Stabilizers in presence of a normal subgroup

If there is a chain of subnormal subgroups this process can be iterated which is the basis of the solvable orbit/stabilizer algorithm, section IV.3.

## Basic Sylow Subgroup Computation

A second 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 **??**) just now are reaching feasibility.

The basic idea of the calculation is that if $\varphi\colon G \to H$ is a homomorphism to a smaller group, we first compute a $p$-Sylow subgroup $S^\varphi \le H$. 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.46: Suppose $G$ is a subdirect product of $A = G^\alpha$ with $B = G^\beta$. Let $Q \le G$ be such that $Q^\alpha$ is a $p$-Sylow subgroup of $A$ and let $\nu = \beta|_Q$ be the restriction of $\beta$ to $Q$. Let $P \le Q$ be such that $P^\nu$ is a $p$-Sylow subgroup of $Q^\nu$. 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 $P^\alpha$ and $P^\beta$ 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 $\varphi$ we can reduce to the situation that $G^{\varphi}$ is a $p$ group, which we will assume now.

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 lies in the center of a Sylow subgroup, or 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 $\Omega$ into orbits of $\langle g\rangle$, it cannot be primitive. If $C = G$, then (by the assumption about $G$) we would have $G$ being imprimitive with blocks corresponding to cycles of $g$. But then the kernel of the block action must fix and centralize ($C = G$!) all $p$-cycles, and therefore is a $p$-group, making $G$ a $p$-group as well, in which case we are done.

We therefore can assume that $C \neq G$ and thus can compute recursively a $p$-Sylow subgroup $S$ of $C$. If $S$ is a Sylow subgroup of $G$ (by order) we are done.

Otherwise, observe that as $g \in Z(C)$, it must lie in every Sylow subgroup of $C$, in particular in $S$. Therefore there is a $p$-Sylow subgroup $P \leq G$, such that $g \in S \leq P$, we thus have that $S = C \cap P$. There must be an element $z \in Z(P)$ of order $p$, i.e. $P \leq C_G(z)$. Because it commutes with $g$, we know that $z \in C \cap P = S$ and clearly $z \in Z(S) \leq Z(P)$.

We thus search for an element of order $p$ in $Z(S)$ for which $C_G(z)$ contains a $p$-Sylow subgroup of $G$. As in the first case we can then recurse on $C_G(z)$.

ALGORITHM II.47: Sylow subgroup computation

```
Input: A group $G$ on $\Omega$ 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 $\Omega$ then
    recurse on orbit actions, using lemma II.46
  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.46
  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 $p \nmid [G:C]$ then
    return $S$;
  fi;
  Let $Z = Z(S)$ {iterative centralizer computation}
  for $z \in Z$, $|z| = p$ do
    $C := C_G(z)$;
    if $p \nmid [G:C]$ 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.48: If $G$ is a group and $n$ 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 this group the *direct power* of $G$ with exponent $m$

Definition II.49: 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 $G \wr H$.

The multiplication in the wreath product is simply given by the rules for a semidirect product. If we consider elements of $G \wr H$ as tuples $(h; g_1, \ldots, g_n)$, we get the following formula:

$$\begin{aligned}
& (h; g_1, \ldots, g_n) \cdot (a; b_1, \ldots, b_n) \\
= \;& (h; \underline{1}) \cdot (1, g_1, \ldots, g_n) \cdot (a; \underline{1}) \cdot (1, b_1, \ldots, b_n) \\
= \;& (h; \underline{1}) \cdot (a; \underline{1}) \cdot \left((a; \underline{1})^{-1} \cdot (1, g_1, \ldots, g_n) \cdot (a; \underline{1})\right) \cdot (1, b_1, \ldots, b_n) \\
= \;& (h \cdot a; \underline{1}) \cdot (1, g_1, \ldots, g_n)^{(a;\underline{\mathbf{1}})} \cdot (1, b_1, \ldots, b_n) \\
= \;& (h \cdot a; \underline{1}) \cdot (1, g_{1^{a^{-1}}}, \ldots, g_{n^{a^{-1}}}) \cdot (1, b_1, \ldots, b_n) \\
= \;& (h \cdot a; g_{1^{a^{-1}}} \cdot b_1, \ldots, g_{n^{a^{-1}}} \cdot b_n)
\end{aligned}$$

The reason for taking as indices the images $1^{a^{-1}}$ under $a^{-1}$ is purely due to the notation: $a$ maps 1 to $1^a$, so after the component-permuting action we get that the element which *was* in position $1^{a^{-1}}$ now ended up in position 1.

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

The next theorem shows that this imprimitive action can be considered to be the "source" of all block systems.

Theorem II.50 (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 $r_j g = \widetilde{g_j} r_{j^g}$. (To simplify notation we will write $j^g$ to indicate the action on $\mathcal{B}$ via $\psi$, i.e. $j^g := j^{(g^\psi)}$.)

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

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

Furthermore, for a suitable labelling of the points, this homomorphism is simply an embedding of permutation groups, i.e. one can consider $G$ as a subgroup of $T^\varphi \wr G^\psi$.

Note II.51: In the context of representation theory $\mu$ is simply the induced representation $\varphi\uparrow^G$. The theorem then is simply the explicit construction of the induced representation.

<u>Proof:</u> We first check the homomorphism property. Suppose that $g, h \in G$, then by definition of $\mu$, we have that

$$\begin{aligned}
g^\mu \cdot h^\mu &= \left(g^\psi; \widetilde{g_{1^{g^{-1}}}}^\varphi, \ldots, \widetilde{g_{n^{g^{-1}}}}^\varphi\right) \cdot \left(h^\psi; \widetilde{h_{1^{h^{-1}}}}^\varphi, \ldots, \widetilde{h_{n^{h^{-1}}}}^\varphi\right)\\
&= \left(g^\psi \cdot h^\psi; \widetilde{g_{(1^{h^{-1}})^{g^{-1}}}}^\varphi \cdot \widetilde{h_{1^{h^{-1}}}}^\varphi, \ldots, \widetilde{g_{(n^{h^{-1}})^{g^{-1}}}}^\varphi \cdot \widetilde{h_{n^{g^{-1}}}}^\varphi\right)\\
&= \left((g\cdot h)^\psi; \widetilde{g_{(1^{(gh)^{-1}})}}^\varphi \cdot \widetilde{h_{1^{h^{-1}}}}^\varphi, \ldots, \widetilde{g_{(n^{(gh)^{-1}})}}^\varphi \cdot \widetilde{h_{n^{g^{-1}}}}^\varphi\right)\\
&= \left((g\cdot h)^\psi; \left(\widetilde{g_{(1^{(gh)^{-1}})}} \cdot \widetilde{h_{1^{h^{-1}}}}\right)^\varphi, \ldots, \left(\widetilde{g_{(n^{(gh)^{-1}})}} \cdot \widetilde{h_{n^{g^{-1}}}}\right)^\varphi\right) \qquad \text{(II.52)}
\end{aligned}$$

by the above multiplication formula. (Again, $h$ is permuting the components via the image $h^\psi$. I.e. the element in position 1 after permutation is what was in position $k = 1^{h^{-1}}$ before, i.e. the element $\widetilde{g_{k^{g^{-1}}}}^\varphi = \widetilde{g_{(1^{h^{-1}})^{g^{-1}}}}^\varphi$.)

We now observe that

$$r_j(g\cdot h) = \widetilde{g}_j r_{j^g} h = \widetilde{g}_j \widetilde{h}_{j^g} r_{(j^g)^h} = \widetilde{g}_j \widetilde{h}_{j^g} r_{(j^{(gh)})}$$

and therefore $\widetilde{(g \cdot h)}_j = \widetilde{g}_j \widetilde{h}_{j^g}$. Setting $j = i^{(gh)^{-1}}$ we get

$$\widetilde{(g \cdot h)}_{i^{(gh)^{-1}}} = \widetilde{g}_{i^{(gh)^{-1}}} \widetilde{h}_{(i^{(gh)^{-1}})^g} = \widetilde{g}_{i^{(gh)^{-1}}} \widetilde{h}_{i^{(h^{-1}g^{-1}g)}} = \widetilde{g}_{i^{(gh)^{-1}}} \widetilde{h}_{i^{h^{-1}}}.$$

This lets us simplify the products in (II.52) to

$$g^\mu h^\mu = \left((g \cdot h)^\psi; \widetilde{(g \cdot h)}_{1^{(gh)^{-1}}}, \ldots, \widetilde{(g \cdot h)}_{n^{(gh)^{-1}}}\right) = (g \cdot h)^\mu,$$

which shows that $\mu$ is a homomorphism.

If $g \in \operatorname{Kern} \mu$ then clearly $g \in \operatorname{Kern} \psi$, implying that $r_j g = \widetilde{g}_j r_j$ and thus $\widetilde{g}_j = r_j g r_j^{-1}$. The values of $\widetilde{g}_j^{\,\varphi}$ that are simply given by the action of $g$ on the multiple blocks. Triviality of all these ensures that $g$ must be the identity.

For the final statement, observe that the transitivity of $G$ on the blocks and of $T$ on its block implies the transitivity of $G^\mu$ on the points moved by the wreath product in its imprimitive action. Furthermore, if $g \in \operatorname{Stab}_G(1)$ then $g^\psi$ fixes the point 1 and $\widetilde{g_{1^{g^{-1}}}}^{\varphi} = \widetilde{g}_1 \varphi$ fixes one point as well. The homomorphism $\varphi$ therefore maps a point stabilizers to a point stabilizer, for transitive groups of the same degree this implies that $\mu$ is a permutation homomorphism. □

Note II.53: 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

Agir en primitif et prévoir en stratège.

Feuillets d'Hypnos #72
René Char

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.50 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

The heart of the analysis will be the consideration of particular normal subgroups. This is motivated by the following corollary from lemma II.45:

Corollary II.54: If $G$ is primitive on $\Omega$, and $\langle 1\rangle \neq N \lhd G$, then $N$ must act transitively on $\Omega$.

We now look at this in the extreme case of the smallest possible $N$.

Definition II.55: A normal subgroup $N \lhd G$ is called minimally normal, if $\langle 1\rangle \neq N$ and there is no normal subgroup $M \lhd G$ such that $\langle 1\rangle \neq M \neq N$ and $\langle 1\rangle < M < N$.

Lemma II.56: Let $G$ be a group and $N \lhd G$ a minimally normal subgroup. Then $N \cong T^{\times k}$ with $T$ simple.

Proof: Let $M \lhd N$ 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 $K := \bigcap_{S\in\mathcal{M}} S \lhd G$, by minimality of $N$ we get that $K := \langle 1\rangle$. Thus $\varphi$ 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 exercise **??**) to a direct product. □

Definition II.57: The *socle* of a group $G$ is the subgroup generated by all minimal normal subgroups:

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

Lemma II.58: $\mathrm{Soc}(G)$ is the direct product of minimal normal subgroups.

Proof: Let $M \le \mathrm{Soc}(G)$ 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 $N \lhd G$, minimally normal, such that $N \not\le M$. But then $M \cap N \lhd G$. As $N$ is minimally normal this implies that $M \cap N = \langle 1\rangle$. Thus $\langle M, N\rangle = M \times N \le \mathrm{Soc}(G)$, 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.59: A permutation group $G$ is *semiregular* on $\Omega$ if $\mathrm{Stab}_G(\omega) = \langle 1\rangle$ for every $\omega \in \Omega$.

Thus $G$ is regular on $\Omega$ if and only if $G$ is transitive and semiregular.

Lemma II.60: Let $G \le S_\Omega$ 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 $\delta$. Thus $c = 1$. □

Lemma II.61: Let $G$ be a primitive group on $\Omega$. Then one of the following situations holds:

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

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

Proof: Suppose that $\mathrm{Soc}(G)$ is not minimally normal. By lemma II.58 we have that $\mathrm{Soc}(G) = N \times M$ with $N \lhd G$ minimally normal and $\langle 1 \rangle \neq M \lhd G$. Then $M \leq C_G(N)$ and $N \leq C_G(M)$.

As $G$ is primitive, $N$ is transitive on $\Omega$. Thus by lemma II.60 we have that $M$ must be semiregular. On the other hand $M \lhd G$ implies that $M$ is also transitive on $\Omega$, 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 $\varphi$ 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 $\varphi$ 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 $T$*.

Definition II.62: Let $G$ be a group, acting on a vector space $V$. (For example, $G \leq \mathrm{GL}_n(p)$ and $V = \mathbb{F}_p^n$.) We say that $G$ acts *irreducibly*, if the only subspaces of $V$ which are invariant under the action of $G$ are $V$ and $\langle 0 \rangle$. (See section **??** for the larger context.)

Theorem II.63: Let $G$ be primitive on $\Omega$ 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, $S \cap \mathrm{Soc}(G) \lhd \mathrm{Soc}(G)$. Also $S \cap \mathrm{Soc}(G) \lhd S$. Thus $S \cap \mathrm{Soc}(G) \lhd G$ 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 $T \lhd \mathrm{Soc}(G)$, thus $T \lhd G$ 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 $\mathbb{F}_p^m$. The combination of a linear action with a translation is called an *affine* action, and the primitive groups with abelian socle are therefore called of *affine type*. They are correspondence with irreducible subgroups of $\mathrm{GL}_n(p)$.

Corollary II.64: Let $G$ be a solvable group and $M < G$ a maximal subgroup. Then $[G{:}M] = p^m$ 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. □

We now study the remaining case, namely that of a nonabelian socle.

Lemma II.65: 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.66: 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.67: 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

In this section we introduce important classes of primitive groups. In view of the preceding corollary, these are in obvious ways subgroups of wreath products.

The first class is a different action of wreath products: Let $G$ be a permutation group on $\Omega$ and $H$ a permutation group on $\Delta$. 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 $\Omega^\Delta$. 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 $\Omega^\Delta$ as a $d$-dimensional cube each side of which is labeled by $\Omega$. We then let $G^{\times d}$ act independently in each dimension and $H$ permute the dimensions. That is, we define

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

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.68: Suppose that $\Omega$, $\Delta$ are finite. Then $G \wr H$ in the product action is primitive if and only if $G$ is primitive, but nonregular on $\Omega$ and $H$ transitive on $\Delta$.

NOTE II.69: 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.70: Let $G \leq S_n$ be a transitive group and $\varphi \in \mathrm{Aut}(G)$. Then $\varphi$ 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 $1 \leq j \leq n$.

Proof: Exercise **??**. □

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 $T^{\times m} \leq G \leq N$ is said to be of *diagonal type*.

THEOREM II.71: 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.72 (O'NAN-SCOTT theorem): Let $G$ be a group which acts primitively and faithfully on $\Omega$ 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", "Holomorph[10] of an abelian group". $T$ is abelian of order $p$, $n = p^m$ and $\mathrm{Stab}_G(\omega)$ is a complement to $H$ which acts irreducibly on $H$.

[10] The *holomorph* of $G$ is the group $G \rtimes \mathrm{Aut}(G)$

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) $m \ge 2$ 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 $V_\omega$ 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 $G \le W = A \wr B$ 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 $n = |T|^m$. $G_\omega$ 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, $m \ge k+1$ where $k$ is the smallest degree of a permutation group which has $T$ as a composition factor.

NOTE II.73: 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.74: 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 |
| [Pra90] | HA | AS | HS | SD | HC | CD | PA | TW |

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.75: 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.76: 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 (i.e. we want the degree to stay the same or become smaller) action – a homomorphism (which we can evaluate by performing the action) $\varphi\colon G \to H$ 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 $\varphi$ yields a composition series.

If $G$ is an intransitive permutation group we can take for $\varphi$ the action of $G$ on one orbit. If $G$ is imprimitive we can take for $\varphi$ 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.77: A transitive permutation group $G$ on $\Omega$ 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 $\beta$ 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 $\Delta$, $N \cap C$ acts regularly on $\Delta$, thus $|\Delta|$ must be a $p$-power (if either of this is not the case, $G$ has no EARNS).

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

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

Let $K \leq C$ such that $K^\varphi = (N \cap C)^\varphi$ is the EARNS of $C^\varphi$. 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 Kern $\varphi \leq Z(G_{\alpha\beta})$ and by elements of $N \cap C$ which commute with each other ($N$ is abelian) and with Kern $\varphi \leq G_{\alpha\beta}$ (as they are in $C$).

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

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

Then $1 \neq x^{\varphi} = g^{\varphi}$ for some $g \in N \cap K \leq$ and $x^{-1}g \in \text{Kern}\,\varphi$. 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.78: A variant of this method can be used to find in a regular normal subgroup of $G$ also for type 5 groups.

Note II.79: Variants of the method (see [LS97]) can be used to find the largest normal $p$-subgroup (also called the *p-core*) $O_p(G) \lhd G$, and from this 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 method given here follows [Neu87]. They differ from what is used in practice but give an idea of the methods used, while being easier to understand.

Theorem II.80 (Schreier's conjecture): Let $G$ be a simple non-abelian group. Then $\text{Aut}(G)/G$ is solvable of derived length $\leq 3$.

Proof: Inspection, following the classification of finite simple groups [Gor82]. □

Lemma II.81: Let $G$ be a primitive group with no nontrivial abelian normal subgroup. Let $S := \text{Soc}(G) = T_1 \times \cdots \times T_m$ 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 $\text{Aut}(T) \wr S_m$ we thus see that elements of $Z(U)$ may not move component $i$. Thus $Z(U) \leq \text{Aut}(T)^m \cap G \lhd G$. Thus $\langle Z(U) \rangle_G \leq \text{Aut}(T)^m \cap G$. But $(\text{Aut}(T)^m)''' = T^{\times m}$ by theorem II.80.

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 $\text{Soc}(G)$.

Note that the only case in which $\text{Soc}(G)$ 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.

## Composition Series

Now we have all tools together to determine a composition series for a permutation group. If a group is primitive, we determine the socle and its direct factors. If the socle is abelian, the group is of affine type. In this case we can take the conjugation action on the nonzero socle elements to get a homomorphism with nontrivial kernel. (Note that the methods of chapter **??** then provide methods for obtaining a composition series through the socle.)

Otherwise, the socle is a direct product of $m$ nonabelian simple groups. If $m > 1$, we can take the conjugation action on these $m$ factors to get a homomorphism. If $m = 1$, or we are in diagonal type 3a, this homomorphism has trivial image. In this case, however, we know that $\left[G \colon \operatorname{Soc} G\right]$ must be small due to Schreier's conjecture II.80. In such a situation we can simply take the regular permutation action for $G/\operatorname{Soc}(G)$ as homomorphism.

Together, this provides reductions until $G$ is simple, we therefore get a composition series.

The same idea can of course be used to test whether a group is simple. For finite simple groups, the classification [Gor82] furthermore provides the information that in most cases the isomorphism type of such a group can be determined from the group's order. In particular, we have the following result of Artin (for classical groups) [Art55] and Teague [Cam81]:

Theorem II.82: Let $G, H$ be finite simple groups with $|G| = |H|$, but $G \ncong H$. Then either (up to swapping the groups)

- $G \cong A_8 \cong \mathrm{PSL}_4(2)$ and $H \cong \mathrm{PSL}_3(4)$.
- $G = \mathrm{PSp}_{2m}(q)$ and $H \cong O_{2m}(q)$ for $m \geq 3$ and odd $q$. (These are the Dynkin diagrams of type $B$ and $C$.)

In either of these two special cases, one can use further, easily obtained, information such as centralizer orders to distinguish the groups.

## 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.83: 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: Exercise **??**. □

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

In practice one would first compute the radical $R := O_\infty(G)$. As this is obtained from iterated computations of $p$-cores $O_p(G)$ we have in fact already some split 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 $G \le GL_n(p)$. Here the action is on vectors in $\mathbb{F}_p^n$.
- Groups of group automorphisms $G \le \mathrm{Aut}(H)$. The action is on elements of $H$.

Note II.84: 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 $G \curlywedge H$ 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

The `genss` package provides a generalized stabilizer chain setup that also will work for example for matrix groups.

### Backtrack

### Blocks and primitivity

The test for primitivity is similar to the orbit functionality, in particular it is possible to provide a domain and an action function.
`IsPrimitive(G,dom,actfun)` tests whether $G$ acts (transitively and) primitively on the domain `dom`. If it does not, `Blocks(G,dom,actfun)` determines a nontrivial block system. (It returns `fail` if the group acts primitively.)
For a permutation group acting transitively on its natural domain, `AllBlocks(G)` returns representatives (namely the blocks containing 1) of all nontrivial block systems.

#### Primitive Groups

Primitive groups have been classified up to order several 1000, and are available via the `PrimitiveGroup` selector function.

### Subdirect products and wreath products

While there is a generic `SubdirectProduct` command, the easiest way to create them is as subgroups of `P:=DirectProduct(A,B)`. Let `e1:=Embedding(P,1)`, `e2:=Embedding(P,1)`. Then `Image(e1,x)*Image(e2,y)` is the element represented by the pair `(x,y)`.
The subdirect product then is generated by first modifying all generators of `A` with a corresponding element of `B`, making the mappings compatible. Using names for greek letters, and constructing the corresponding elements of the the command:

```
List(GeneratorsOfGroup(A),
     x->Image(e1,x)
       *Image(e2,PreImagesRepresentative(sigma,
                         Image(zeta,Image(rho,x))  )));
```

together with (Image(e2,E)) (which is the kernel of `beta` as subgroup of the direct product).
The wreath product $G \wr H$ can be created as `WreathProduct(G,H,a)`, where $a \colon H \to S_n$ is a homomorphism indicating the permutation action. If $H$ is a permutation group itself, the homomorphism can be ommitted.
If both $G$ and $H$ are permutation groups, `WreathProductImprimitiveAction` and `WreathProductProductAction` create particular permutation actions.
For any product `W` created this way, `Embedding(W,i)` for $1 \leq i \leq n$ are monomorphisms $G \to W$, `Embedding(W,n+1)` is the monomorphism $H \to W$. Finally `Projection(W)` is the epimorphism $W \to H$.

## Composition series and related functions

Series for a group are always returned as a list, descending from larger to smaller subgroups. `CompositionSeries(G)` determines a composition series. `DisplayCompositionSeries(G)` prints a compositylon series structure in human-readable form.
Based on comnposition series methods, `ChiefSeries(G)` calculates a chief series. `ChiefSeriesThrough(G,normals)` determines a series through particular normal subgroups. `ChiefSeriesUnderAction(H,G)` returns a series of subgroups normal under the action of the supergroup `H`.

## 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, $a, b, c, \ldots$ — in this case often upper case letters are used to denote inverses.) 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 $\sim$ as the equivalence relation (smallest classes) on $A^*$ induced by direct equivalence. We now consider $F = A^*/\sim$. 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 $\underline{g}$ is a free generating set for $F$ we write $G \cong \left\langle \underline{g} \mid R \right\rangle$ 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 $\varphi\colon F_{|G|} \to G$ that maps generators to the elements of $G$. It extends to a surjective homomorphism. Kern $\varphi$ 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 representatives are not unique. Also there is in general no easy "normal form" as there is for small examples. (See chapter V 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 $G = \left\langle \underline{g} \mid R \right\rangle$. 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 $\underline{g}$, add a new generator $x$ to $\underline{g}$ and a new relator $x^{-1}w$ to $R$
4. If a generator $x \in \underline{g}$ 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. □

Note that steps can be combined to have one relator be used to reduce another one.

Tietze transformations were defined in the context of the following

Lemma III.11: Suppose that the presentations $P_1 = \left\langle \underline{g} \mid R \right\rangle$ and $P_2 = \langle \underline{h} \mid S \rangle$ 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 $Q = \left\langle \underline{g} \cup \underline{h} \mid R \cup S \cup T \right\rangle$ by adding relators $T$ that express $\underline{h}$ in terms of $\underline{g}$ and deduce the relators in $S$, then delete the redundant $\underline{g}$ by expressing them as words in $\underline{h}$ 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 $x \in \underline{g}$. 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 $x \in \underline{g}$ 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 (von Dyck's Theorem): Let $G = \left\langle \underline{\boldsymbol{g}} \mid R \right\rangle$ be a finitely presented group. For a group $H$ we define a map $\varphi\colon \underline{\boldsymbol{g}} \to H$. Then $\varphi$ 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: $r(\underline{\boldsymbol{g}}^{\varphi}) = 1$.

Proof: Homework. □

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

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

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

- The relators evaluated in the generator images are trivial: $r(\underline{\boldsymbol{g}}^{\varphi}) = 1_H$, and
- The generator images generate $H$: $H = \left\langle \underline{\boldsymbol{g}}^{\varphi} \right\rangle$ (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 $\varphi$ 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* $(\varphi, U)$ of $G$ is a subgroup $S \le G$ that is given as preimage $S = \varphi^{-1}(U)$ of a subgroup $U \le G^\varphi$ 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 (see exercise **??**):

- $g \in S$ 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 $S, T \le G$ are both quotient subgroups given by the homomorphisms $\varphi$ and $\psi$ respectively, we can consider the larger quotient $\xi\colon G \to G^\varphi \curlywedge G^\psi$ and calculate the intersection there.

If we have a quotient subgroup $S = \varphi^{-1}(U) \le G$ 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 the real world, we will also encounter 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, constructing new cosets as images of existing cosets onder group generators or their inverses. 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 under this relator (as it represents the identity element).

Finally we keep a similar table for every subgroup generator, here however we can only require the trivial coset to remain fixed, as other cosets do not need to remain fixed under the subgroup.

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

| | $a$ | $a^{-1}$ | $b$ | $b^{-1}$ |
|---|---|---|---|---|
| 1 | | | | |

the relator tables initially are:

| $a$ | $a$ |
|---|---|
| 1 | 1 |

| $b$ | $b$ | $b$ |
|---|---|---|
| 1 | | 1 |

| $a$ | $bababab a$ | $b$ |
|---|---|---|
| 1 | | 1 |

and the subgroup tables:

| | $a$ |
|---|---|
| 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 | $\underline{\mathbf{1}}$ | **1** | | |

We also update the other tables:

| | $a$ | $a$ |
|---|---|---|
| 1 | **1**! | 1 |

| | $b$ | $b$ | $b$ |
|---|---|---|---|
| 1 | | | 1 |

| | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **1** | | | | | | | | | 1 |

| | $a$ |
|---|---|
| 1 | 1 |

| | $b^{-1}$ | $a$ | $b$ |
|---|---|---|---|
| 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 | $\underline{\mathbf{2}}$ | |
| 2 | | | | **1** |

| | $a$ | $a$ |
|---|---|---|
| 1 | 1 | 1 |
| 2 | | 2 |

| | $b$ | $b$ | $b$ |
|---|---|---|---|
| 1 | **2** | | 1 |
| 2 | | **1** | 2 |

| | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | **2** | | | | | | | | 1 |
| 2 | | | | | | | | **1** | **1** | 2 |

| | $b^{-1}$ | $a$ | $b$ |
|---|---|---|---|
| 1 | | | 1 |

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

| | $a$ | $a^{-1}$ | $b$ | $b^{-1}$ |
|---|---|---|---|---|
| 1 | 1 | 1 | $\underline{2}$ | $\underline{\mathbf{3}}$ |
| 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 |

| | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 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 | $\underline{2}$ | $\underline{3}$ |
| 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 | $\underline{2}$ | $\underline{3}$ |
| 2 | $\underline{\mathbf{4}}$ | | 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 | $\underline{2}$ | $\underline{3}$ |
| 2 | $\underline{4}$ | **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 | $\underline{2}$ | $\underline{3}$ |
| 2 | $\underline{4}$ | 4 | 3 | 1 |
| 3 | 3 | 3 | 1 | 2 |
| 4 | 2 | 2 | $\underline{\mathbf{5}}$ | |
| 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 | $\underline{2}$ | $\underline{3}$ |
| 2 | $\underline{4}$ | 4 | 3 | 1 |
| 3 | 3 | 3 | 1 | 2 |
| 4 | 2 | 2 | $\underline{5}$ | $\underline{\mathbf{6}}$ |
| 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 | $\underline{2}$ | $\underline{3}$ |
| 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** |

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, $[G{:}S] = n$ 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 $R^c_g$ starting at the coset $y$ and all relators in $R^c_{g^{-1}}$ 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 | 1 | |
| 2 | 3 | 2 | |
| 3 | 2 | 3 | |
| 4 | 6 | 4 | |
| 5 | | 5 | |
| 6 | 4 | 6 | |

| | $b$ | $b$ | $b$ |
|---|---|---|---|
| 1 | 2 | | 1 |
| 2 | | 1 | 2 |
| 3 | 4 | 5 | 3 |
| 4 | 5 | 3 | 4 |
| 5 | 3 | 4 | 5 |
| 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 |

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

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

THEOREM III.24 (Mendelsohn, 1964): Suppose that $[G:S] < \infty$ 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 $[G:S] < \infty$. □

### 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.

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 $G = \langle \underline{\mathbf{g}} | R \rangle$ 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: $G = \langle \underline{\mathbf{G}} \mid R \rangle$, 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 $U \le Q$ of small index. (Here we use the knowledge of groups of small order to obtain concrete bounds.)

Then the preimage of $U$ under $\varphi$ can be obtained by an ordinary low-index calculation for such a small index, together with quotient algorithms applied to subgroups.

A detailled algorithm doing this is described in [Fir05] and e.g. implemented in GAP with the LINS package.

An somewhat alternative view of the low-index algorithm is 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 main benefit of the low-index routine is that it implicitly uses the defining relators to impose conditions on the permutations. This can be of advantage, if the quotient group is large (which typically means: $S_n$ or $A_n$).

## 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 $G = \left\langle \underline{\boldsymbol{g}} \mid R \right\rangle$ as a quotient of $F$. We consider $\underline{\boldsymbol{g}}$ as the elements of $G$ that are the images of the free generators $\underline{\boldsymbol{f}}$.

Suppose that $S \leq G$ with $n = [G{:}S] < \infty$. We choose a transversal of coset representatives for $S$: $t_1 = 1, t_2, \ldots, t_{[G:S]}$, 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 representative $t_i$ for the $i$-th coset is defined as $t_i = t_j \cdot g_x$, the Schreier generator $s_{j,x}$ is trivial by definition. This is exactly the case when we make a new definition in the coset table. In this case, 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 $\rho\colon U \to E$ 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 $G = \langle \underline{g} \mid R \rangle$ a finitely presented group and $S \leq G$ with $[G{:}S] < \infty$. Then $S$ is finitely presented and

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

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 $\rho$, 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.

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.

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.

Scanning a word in the generators of $G$ through the coset table, and forming the product of the Schreier generators encountered "on the way" then expresses this element as a product of the Schreier generators. (This is exactly what we did in the proof of Schreier's Lemma. The augmented coset table thus provides a nicer way of visualizing this proof.

Reidemeister's theorem then simply states that we need to scan every relator for $G$ at every coset of $S$ to get a presentation in the Schreier generators.

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 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 | $i5$ | $g^{-1}5$ | $4$ | $h^{-1}5$ |

Note that we only define Schreier generators for the generators, and not for inverses. If, for representatives $t_i$, $t_j$ and a generator $g$, we have that $t_i \cdot g = s \cdot t_j$, then $s^{-1}t_i = t_j g^{-1}$. That is the inverse of a Schreier generator is exactly the Schreier generator for the inverse entry in the coset table.

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.

Performance III.37: 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.

## The modified Todd-Coxeter algorithm

In the Reidemeister-Schreier algorithm Schreier generators were introduced in an existing coset table to reflect the difference between representative images and representatives of the images.

A similar result arises if we demand in general that the tables do not just represent equality of cosets, but of group elements. This is done in a variant of coset enumeration, called the *Modified Todd-Coxeter* algorithm (MTC). In it, difference between images and representatives is corrected by augmenting tables with suitable products of subgroup generators.

For this, introduce symbols $u_i$ that represent the subgroup generators. In the $i$-th (one row) subgroup table, we then have that the image of coset 1 is $u_i \cdot 1$ (since it reflects the tracing of the $i$-th subgroup generator. The relator tables however (which all represent the identity of the group) remain with rows starting and ending with a number $j$. We also reserve space in the coset table to store correcting factors, given as products of the subgroup generators.

The general process than follows exactly as in the ordinary coset enumeration, with the difference that we maintain equality of elements. Thus:

- When closing a table row – say we have that coset x under generator g gives coset y – there might be factors associated with the two entries that join, that is the entries are actually $a \cdot x$ and $b \cdot y$ with $a$ and $b$ words in the $\{u_i\}$. Then the consequence we record in the coset table is that the image of $x$ under $g$ is $a^{-1}b \cdot y$ and the image of $y$ under $g^{-1}$ is $b^{-1}a \cdot x$. This introduces *cofactors* (words in the $\{u_i\}$) into the augmented coset table.

- When recording images from the coset table in the subgroup or the relator table, these cofactors are kept. That is, if we have entry $a \cdot x$ in a table and know, from the coset table, that $x$ under the next generator $g$ maps to $b \cdot y$, we enter $ab \cdot y$ in the next column of the table.

Instead of describing this in a formal algorithm, we illustrate this by re-doing the same coset enumeration as in Example III.20, just this time with the extra cofactors.

Example III.38: We call the subgroup generators $s = a$ and $t = a^b$ and thus have the subgroup tables as:

| | $a$ | |
|---|---|---|
| 1 | | $s \cdot 1$ |

| | $b^{-1}$ | | $a$ | | $b$ | |
|---|---|---|---|---|---|---|
| 1 | | | | | | $t \cdot 1$ |

From the first table, we deduce that $1^a = s \cdot 1$ and enter this in the coset table:

| | $a$ | $a^{-1}$ | $b$ | $b^{-1}$ |
|---|---|---|---|---|
| 1 | $\mathbf{s \cdot 1}$ | $\mathbf{s^{-1} \cdot 1}$ | | |

(Entering this in the first row of the relator table for $a^2$ would give us that $s^2 = 1$, but we don't need to record this now since it will arise from relator tracing.)

Next we define $2 = 1^b$ as before with no change to the previous entries. Once we define $3 = 1^{b^{-1}}$, however we need to enter this correctly into the second subgroup table

We also get the second subgroup table as:

| $b^{-1}$ | $a$ | $b$ |
|---|---|---|
| 1 | $\mathbf{3!t\cdot 3}$ | $t\cdot 1$ |

This gices the consequences that $3^a = t\cdot 3$, and $3^{a^{-1}} = t^{-1}\cdot 3$. We enter this in the coset table.

The definitions of $4 = 2^a$, $5 = 4^b$ and $6 = 4^{b^{-1}}$ again proceed as before. At this point, the first row of the relator table for $(ab)^5$ closes:

| $a$ | $b$ | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ | $a$ | $b$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | $s\cdot 1$ | $s\cdot 2$ | $s\cdot 4$ | $s\cdot 5!\mathbf{t^{-1}\cdot 6}$ | $t^{-1}\cdot 4$ | $t^{-1}\cdot 2$ | $t^{-1}\cdot 3$ | 3 | 1 |
| 2… | | | | | | | | | |
| 3… | | | | | | | | | |

this gives the cosequence $5^a = s^{-1}t^{-1}\cdot 6$ and $6^{a^{-1}} = ts\cdot 5$. Entering this in the relator table for $a^2$ gives the consequences that $6^a = ts\cdot 5$ and $5^{a^{-1}} = s^{-1}t^{-1}\cdot 6$. The augmented coset table now is complete as:

| | $a$ | $a^{-1}$ | $b$ | $b^{-1}$ |
|---|---|---|---|---|
| 1 | $s\cdot 1$ | $s^{-1}\cdot 1$ | $\underline{2}$ | $\underline{3}$ |
| 2 | $\underline{4}$ | 4 | 3 | 1 |
| 3 | $t\cdot 3$ | $t^{-1}\cdot 3$ | 1 | 2 |
| 4 | 2 | 2 | $\underline{5}$ | $\underline{6}$ |
| 5 | $s^{-1}t^{-1}\cdot 6$ | $s^{-1}t^{-1}\cdot 6$ | 6 | 4 |
| 6 | $ts\cdot 5$ | $ts\cdot 5$ | 4 | 5 |

As before, we can now trace words in the generators of $G$ through the coset table, accumulating a word in the generators of $S$ through the cofactors.

If we trace a word that represents an element $g\in S$ from coset 1, we need to end at coset 1 again. The fact that the coset table reflects euqality of elements, not just of cosets, however means that the collected cofactor word is an expression for $g$ as a word in the generators of $S$.

If we compare this augmented coset table with the one in Example III.35 (page 80), we note that the cofactors express the nontivial Schreier generators as words in the chosen generators of $S$. It is not hard to show that this holds in general.

We thus get the obvious corollary of Theorem III.30: Traciong all relators at all cosets yields a presentation for $S$ *as a finitely presented group in the chosen ghenerators!*

Example III.39: We continue the example. Tracing $a^2$ at coset 1 yields (as promised) the traced word (and thus relator) $s^2$. Ditto we get $t^2 = 1$ from tracing relator $a^2$ at coset 3. Tracing $a^2$ at coset 5 gives $s^{-1}t^{-1}ts$, which is the trivial word, and thus no useful relator. Similarly almost all other traces yield relators that are trivial or cancel freely. Tracing coset 6 through relator $(ab)^5$ however (finally) yields the word $(st)^5$.

We conclude, that

$$S = \left\langle s, t \mid s^2 = t^2 = (st)^5 \right\rangle$$

is a presentation for $S$ (and thus $S$ is a dihedral group of order 10).

Note III.40: This example is misleading in that it suggests the MTC to produce a nicer subgroup presentation, furthermore on a chosen (smaller) generator set. This is misleading: Once the index gets larger the length of the cofactor expressions (iterated products double length with each multiplication) often get infeasibly long. This is even an issue for the coset enumeration itself – algorithms typically store these cofactors in a *decoding tree* that reflects the multiplication structure, storing each factor as a product/quotient of prior factors (and thus of shorter length). Furthermore, Tietze transformations (see III.34) typically are better at reducing the generator number of a presentation onbtained by Reidemeister-Schreier, than reducing the relator lengths obtained from MTC. MTC however is the way to go for evaluating homomorphisms, given on subgroup generators (which are not Schreier generators chosen for an existing coset table).

An important application of MTC is the following method, often used to determine the order of a finite finitely presented group: Choose for $S$ a cyclic subgroup $\langle x \rangle$ (often $x$ is itself chosen as a generator of the group or a short product of generators). In this case all cofactors will be powers of $x$ (and thus the issue of word length is not critical). Furthermore, all relators for $s$ obtained will have the form $x^{e_i} = 1$, thus $|\langle x \rangle| = \mathrm{lcm}_i(e_i)$ can be obtained easily. One then has $|G| = \left[G \colon \langle x \rangle\right] \cdot |\langle x \rangle|$.

## 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 V.6.

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 $F/N \cdot \mathrm{Kern}\,\varphi \cong G/G'$.

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 $\pm 1$): 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 $\infty$ 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.41: 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.

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.

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

## 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 V.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 generating 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. This is in fact an alternate view of problem **??** in chapter II.

Note III.42: 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.11.

## Using the composition 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 will get a presentation in a new generating set. In most applications this is of little concern.

By induction it is sufficient to describe a process that constructs a presentation for a group $G$ from a presentation for $N \lhd G$, one for $G/N$, and a choice of elements of $G$ that represent the generators for $G/N$.

Lemma III.43: 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.44: 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.

### 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

$$\left\langle g_1,\ldots,g_{n-2} \mid \forall i, j > i : g_i^3 = (g_i g_j)^2 = 1 \right\rangle$$

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 **??**.

Alas very little of this approach is actually implemented.

## III.11 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 **??**), write down a presentation for each simple factor $F$.
4. Use the method of lemma III.43, 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.

## III.12 How to do it in GAP

### Free Groups and Finitely Presented Groups

Free groups are generated either for a particular rank (as `FreeGroup(`$k$`)` for an integer $k$, or for particular names of the generators as `FreeGroup(`$s_1$`,`$s_2$`,...,`$s_k$`)` for strings $s_i$. Note that the names of the generators are purely print names and do not necessarily correspond to variables[3], however the convenience function `AssignGeneratorVariables(`$F$`)` when applied to a free group $F$, will assign – if possible – global variables to the group generators named that way, overwriting previous values of these variables.

[3] For connoisseurs of the confusing it is possble to have different generators being named the same, or to assign generator "x" to variable "y" and vice versa.

Finitely presented groups are created as quotiets of a free group by a list of relators, relators being expressed as elements of the free group. If $F$ is a free group whose generators are all named by single letters, the function `ParseRelators(F,s)` will take a string $s$ which represents a presentation written in traditional form – possibly including equalities, and return a list of relators of an equivalent presentation. Numbers (even without a leading caret operator) are interpreted as exponents, a change from lower- to upper-case (or vice versa) is taken to represent inverses. Thus, the following three definitions all define the same finitely presented group (of order 18), differing only in the level of convenience to the user:

```
gap> f:=FreeGroup("a","b","c");
<free group on the generators [ a, b, c ]>

gap> AssignGeneratorVariables(f);
#I  Assigned the global variables [ a, b, c ]
gap> rels:=[f.1^f.2/f.3,f.3*f.1/(f.2^-1*f.3),
> Comm(f.1,f.2^2*f.3), (f.2^3*f.3^-1)^2*f.1^-10];
[ b^-1*a*b*c^-1, c*a*c^-1*b, a^-1*c^-1*b^-2*a*b^2*c,
  b^3*c^-1*b^3*c^-1*a^-10 ]

gap> rels:=[a^b/c,c*a/(b^-1*c),Comm(a,b^2*c),(b^3*c^-1)^2/a^10];
[ b^-1*a*b*c^-1, c*a*c^-1*b, a^-1*c^-1*b^-2*a*b^2*c,
  b^3*c^-1*b^3*c^-1*a^-10 ]

gap> rels:=ParseRelators(f,"a^b=c,Bc=ca,[a,b2c],(b3C)2=a10");
[ b^-1*a*b*c^-1, c*a*c^-1*b, a^-1*c^-1*b^-2*a*b^2*c,
  b^3*c^-1*b^3*c^-1*a^-10 ]
gap> g:=f/rels;
<fp group on the generators [ a, b, c ]>
```

## Elements of Finitely Presented Groups

The elements of a finitely presented group $G$ are displayed the same way as the elements of the (underlying) free group, but they are different objects. (The function `AssignGeneratorVariables` will also work for finitely presented groups. In particular, equality tests for elements of a finitely presented group will potentially trigger hard (or even impossible) calculations to establish for example a faithful permutation representation. This however does not impl that elements are automatically reduced, as the following example shows:

```
gap> f:=FreeGroup("a");
<free group on the generators [ a ]>
gap> g:=f/[f.1^5];
<fp group on the generators [ a ]>
```

```
gap> a:=g.1;;
gap> a^5; # display is not reduced
a^5
gap> a^5=a^0; # but test is properly done
true
```

If $G$ is small or has a well-behaving[4] presentation, the command `SetReducedMultiplication(`$G$`)` will enforce an intermediate reduction of products, representing each newly created element by a normal form representative

```
gap> SetReducedMultiplication(g);
gap> a^5;
<identity ...>
gap> a^28;
a^-2
```

## Creating and Decomposing Elements

For an element $x$ of a finitely presented group $G$, `UnderlyingElement(`$x$`)` returns the element $y$ of the free group used to represent $x$. For an element $y$ of a free group, `LetterRepAssocWord(`$y$`)` returns a list of integers that represents $y$ as a product of the (numbered) generators, negative numbers indicating inverses. The reverse translations require *element families* (which are system objects representing the groups elements without being a group or having an element test). Elements of free groups are created from integer lists by `AssocWordByLetterRep(`*familyfree*`,`*intlist*`)`, elements of finitely presented groups via `ElementOfFpGroup(`*familyfp*`,`*freeelement*`)`. For example:

```
gap> famfree:=FamilyObj(One(f));;
gap> AssocWordByLetterRep(famfree,[1,2,3,-2,1]);
a*b*c*b^-1*a
gap> famfp:=FamilyObj(One(g));;
gap> ElementOfFpGroup(famfp,f.1*f.2);
a*b
```

A finitely presented group has the attributes `FreeGroupOfFpGroup`,`FreeGeneratorsOfFpGroup`, and `RelatorsOfFpGroup` which let an algorithm obtain the underlying presentation.

[4]well-behaving being defined only in that this functionality works

## Presentations and Tietze Transformations

Presentations are objects that are not groups and which can be modified. `PresentationFpGroup` creates a presentation from a finitely presented group, `FpGroupPresentation` creates a (new!) finitely presented group from a presentation.
For a presentation $P$, the command `TzGoGo(`$P$`)` applies (in a greedy algorithm) Tietze transformations to eliminate generators and make the presentation shorter. (It will only delete generators and never create new ones.) As generator elimination in general increases the length of a presentation, this typically returns a somewhat heuristic result that balances generator number and presentation length. (It clearly is not guaranteed that the result in any way is "shortest" or "standardized".)
The function `IsomorphismSimplifiedFpGroup` does similar, but does the group/presentation translation automatically and also preserves the connection between the old group and the new one.

## Subgroups and Coset Tables

### Subgroups

Subgroups of finitely presented groups typically are using the method of section III.5 for calculations – the actual homomorphism being hidden from the user. It can be obtained via the attribute `DefiningQuotientHomomorphism(`$S$`)`. Subgroups represented this way often get printed as `Group(<fp, no generators known>)` unless the user requested an explicit ist of (Schreier) generators.
If a subgroup is given by generators, typically a coset enumeration is performed to get the permutation representation on the cosets.

### Coset Enumeration

On a pure user level coset enumeration is hidden from the user and performed automatically for purposes such as element tests or group order calculations. One can see coset enumeration being performed by setting the appropriate `InfoLevel` higher: `SetInfoLevel(InfoFpGroup,2)`. Doing so will print status messages about the progress of the enumeration.
To perform a coset enumeration for a subgroup $S \leq G$ on its own, the command `CosetTable(`$G$`,`$S$ can be used. The function returns (for memory reasons, as there are typically far more cosets than generators) a transposed coset table with rows corresponding to the generators and their inverses $g_1$, $g_1^{-1}$, $g_2$ $g_2^{-1}$ and so on, and columns corresponding to the cosets.
On a lower level, `CosetTableFromGensAndRels(`*`fgens`*`,`*`rels`*`,`*`sub`*`)` takes as *`fgens`* the generators of the free group via which $G$ was defined (`FreeGeneratorsOfFpGroup(`$G$`)`), *`rels`* the relators as words in these

generators (`RelatorsOfFpGroup(`$G$`)`), and *sub* generators of the subgroup $S$ *as elements of the free group* (i.e. use `UnderlyingElement`).
Coset tables in GAP typically are standardized using `StandardizeTable`.
The implementation of the Todd-Coxeter algorithm in GAP is relatively basic, the package ace provides an interface to a far more powerful external enumerator.

**Low Index**

The function `LowIndexSubgroupsFpGroup(`$G$`,`$n$`)` determines all subgroups of a finitely presented group $G$ of index $\leq n$ up to conjugacy. In many cases $n \in [10..30]$ is a reasonable index if $G$ is generated by 2 or 3 elements. As the runtime grows with the number of generators, it is usually worth to simplify the presentation first using Tietze transformations.
As low index calculations can take a long time, and it often is useful to get some subgroups quicker, an alternative user interface uses *iterators*: `LowIndexSubgroupsIterator(`$G$`,`$n$`)` creates an iterator object $I$. For this object `NextIterator(`$I$`)` always produces the next subgroup until `IsDoneIterator(`$I$`)` returns `true`.

```
gap> it:=LowIndexSubgroupsFpGroupIterator(g,15);
<iterator>
gap> NextIterator(it);
Group(<fp, no generators known>)
gap> while not IsDoneIterator(it) do
> Print(IndexInWholeGroup(NextIterator(it)),"\n");
> od;
2
3
6
9
```

A variant of the low-index process lets the user provide a subgroup $S$ as third argument – the computer then determines only subgroups containing $S$.

**Subgroup Presentations**

The commands `PresentationSubgroupRrs` (Reidemeister-Schreier, selecting Schreier Generators) and `PresentationSubgroupMtc` (Modified Todd Coxeter, using the provided subgroup generators, see note /refmtc) create a subgroup presentation using an augmented coset table. They return a presentation object, as used by Tietze transformations and indeed immediately apply Tietze transformations to reduce the presentation created and to eliminate obvious redundancies.
A more user friendly interface is by creating homomorphisms, that also provide for translation between subgroup elements in the original group, and elements of

the group in the new presentation. For a subgroup $S$ of a finitely presented group, `IsomorphismFpGroup(`$S$`)` creates such an isomorphism using Reidemeister Schreier, `IsomorphismFpGroupByGenerators(`$S$`,`*gens*`)`, for a generator list *gens*, uses the Modified Todd-Coxeter to produce a presentation on particular desired generators.

## Homomorphisms

### Epimorphisms to known finite groups

The command `GQuotients(`$G$`,`$Q$`)` classifies epimorphisms $G \to Q$, assuming that $Q$ is a group for which conjugacy classes and double cosets can be computed efficiently – e.g. a permutation group. (The operation works not only for a finitely presented $G$, but also for arbitrary finite $G$, effectively computing a presentation for $G$ to test for homomorphisms.)

### Abelian Quotients

For a matrix with integer entries, `SmithNormalFormIntegerMat(`$M$`)` determines the Smith Normal Form, `SmithNormalFormIntegerMatTransforms(`$M$`)` returns a record that also contains transforming matrices such that $S.rowtrans \cdot M \cdot S.coltrans = S.normal$.

`MaximalAbelianQuotient(`$G$`)` forms the corresponding homomorphism. If $G/G'$ is finite the image is a pc group, if it is infinite the image is a finitely presented group.

### Getting Presentations

In consistence with subgroup presentations, for a finite group $G$ the function `IsomorphismFpGroup(`$G$`)` determines a presentation (it returns an isomorphism to the corresponding finitely presented group, from which the presentation can be obtained. The presentation is built using a composition series, generators are chosen by the algorithm.
Similarly `IsomorphismFpGroupByGenerators(`$G$`,`*gens*`)` uses the reverse Todd-Coxeter process to determine a presentation for $G$ on a given generating set.

Chapter

# IV Pc Groups

Solvable groups occupy a special place in group theory: They can be obtained as iterated extensions of cyclic groups – the most basic groups which are – and hold many special properties, both in terms of the existence of structures (such as Sylow systems), and computability (such as solvability of polynomials in terms of radicals). In this chapter we will be concerned less with the particular theory of solvable groups, but will use their special structure simply to obtain a very memory efficient way of storing elements. We will revisit this structure in later chapters, both as a tool for finding quotients of finitely presented groups (Section V.6), as well as for reducing calculations in larger groups to their constituent factors (Section **??**).

Much of what is described in this chapter can be generalized to the case of infinite groups, the reader is referred to [Eic01].

While many algorithms have been developed specifically for solvable groups, they often can be considered as special cases of the more general framework we will describe in chapter **??**.

## IV.1 Pc presentations

Our starting point is that of presentations for extensions as given in section III.10. Assume that $G$ is finite and solvable. Then the composition series consists of cyclic factors of prime order and we trivially get a presentation $\left\langle g \middle| g_p = 1\right\rangle$ for each of these cyclic factors. Combining these, we therefore get a presentation of the following form for $G$:

- 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 IV.1: 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 change the order of generators in a word; 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 $0 \le e_i < p_i$. We call this form the *normal form* of the word, the process of bringing a word into normal form is called *collection* and will be described in more detail in section IV.2 below.

More generally, we can consider such a set of generators $g_1, \ldots, g_n$ for any solvable group $G$. It is called a *polycyclic generating set* (short *PCGS*). Whenever a PCGS is chosen, we get a bijection between group elements and valid exponent vectors.

The composition series defined by suffixes $\langle g_i, g_{i+1}, \ldots, g_n\rangle$ of a PCGS is called the corresponding *PC series*. Given a solvable group there typically is a choice amongst multiple series (and generators) and it is natural to pick a series that has particular desired properties. For example one might want it to refine a chief series. We will study a particularly nice version in section IV.4.

If for $M = \langle g_i, g_{i+1}, \ldots, g_n\rangle$ and $N = \left\langle g_j, g_{j+1}, \ldots, g_n\right\rangle$ we have that $M, N \lhd G$ and $M/N$ is elementary abelian, we call $g_i, g_{i+1}, \ldots, g_{j-1}$ a *layer* in the PCGS.

As in Linear algebra with coefficient vectors it is tempting to translate arithmetic in a solvable group with a chosen PCGS to exponent vectors. Given a full PC presentation this is possible – the product of two elements is obtained simply by concatenating the corresponding words, and collecting the result to normal form.

This makes it possible to use a polycyclic presentation to *define* a solvable group on the computer, representing its elements by exponent vectors. We shall call groups in this representation *PC groups*.

Such groups have a couple of algorithmically interesting properties:

- The storage of elements is very space efficient and close to optimal, being essentially of size $\log |G|$.
- The natural homomorphism for the factor groups related to the composition series is easily evaluated (we just need to trim exponent vectors to a suitable prefix).
- For elementary abelian layers in the series, the corresponding part of the exponent vector yields a vector space representation. This is the principal idea behind many efficient algorithms we will see later (see chapter **??**).

There are essentially three ways of creating such PC groups:

- It may be possible to simply write down the pc presentation for a group of interest (for example, it is not hard to construct the pc-presentation for a semidirect product from pc-presentations for the factors). The reader however should be aware that not any arbitrary presentation that syntactically fulfills the rules for a pc presentation corresponds to a group with that structure. We will find a criterion for this in section V.4.

- If we have already a solvable group, represented in a different way (for example as a permutation group), and choose a PCGS from a composition series, we can determine (as in section III.10) the corresponding PC presentation by evaluating the left sides of the rules of type $R_3$ and $R_4$, and expressing these as words in normal form. We will see a particular efficient way for this in section IV.5.

- Generalizing the idea of abelian quotients from section III.9 there are algorithms that will determine quotients of finitely presented groups in the form of PC groups, see section V.6.

## IV.2 Collection

### A word about complexity

## IV.3 Induced pc systems

Suppose that $G$ is a pc group and $S \leq G$. Suppose we have some generating set $\underline{s} = \{s_1, \ldots, s_k\}$ for $S$. Consider the exponent vectors for the $s_i$ as rows in a matrix. Our aim is to bring this matrix into row echelon form.

Suppose that $s_i$ corresponds to an exponent vector $[a, \ldots]$. Then a power $s_i^e$ will have exponent vector $[1, \ldots]$ (as $\gcd(a, p) = 1$ for $p$ being the relative order in the first component).

Similarly, if $s_j$ has coefficient vector $[b, \ldots]$, then $s_j/s_i^{b/a}$ has exponent vector $[0, \ldots]$.

It is easily seen that these transformations do not change the group generated.

We can therefore transform $\underline{s}$ to a new generating set $\underline{\hat{s}}$ such that the matrix of exponent vectors is in row echelon form. Such a generating set $\underline{\hat{s}}$ is called an *induced generating set* (or *IGS*) for $S$. If we assume reduced row echelon form, we can even obtain a unique generating set for $S$, called a *canonical generating set* (or *CGS*).

If we have an induced generating set for $S$ and $g \in G$ arbitrary, we can attempt to divide generators of the IGS off $g$ to transform the exponent vector for $g$ in the zero vector. This will succeed if and only if $g \in S$. The remainder will be a "canonical" (with respect to the chosen IGS) coset representative for the *left coset* $gS$. (We have to divide off from the right to avoid commutators with remaining higher terms causing problems.)

### Orbit/Stabilizer computations in solvable groups

As suffixes of a PCGS define a subnormal chain, the ideas of section II.4 provide a very elegant way of determining orbits and stabilizers in solvable groups. By induction we only need to consider the case that $g$ is the first element in a PCGS for $G = \langle N, g\rangle$ with $N \lhd G$ generated by the first suffix, and that for $\omega \in \Omega$ we already computed the orbit $\omega^N$ and an IGS for $\mathrm{Stab}_N(\omega)$. As $p = [G{:}N]$ is prime, the following two possibilities are the only possible:

**a)** $|\mathrm{Stab}_N(\omega)| = |\mathrm{Stab}_G(\omega)|$ Then stabilizers stay the same (i.e. we keep the existing IGS) and we extend the orbit to length $p \cdot |\omega^N| = |\omega^G|$ by adding images of $\omega^N$ under $g, g^2, \ldots, g^{p-1}$.

**b)** $|\mathrm{Stab}_N(\omega)| < |\mathrm{Stab}_G(\omega)|$ Then $|\omega^N| = |\omega^G|$, i.e. the orbits stay the same. We determine $n \in N$ such that $(\omega^g)^n = \omega$ and add $g \cdot n$ to the IGS for $\mathrm{Stab}_N(\omega)$. As $p = [\,\mathrm{Stab}_G(\omega){:}\,\mathrm{Stab}_N(\omega)]$ this will be an IGS for $\mathrm{Stab}_G(\omega)$.

## IV.4 LG series

A particular nice situation is that of so-called "special pc-groups" (defined by a series of rather technical conditions [CELG04]), which not only provide particularly good algorithms, but also often a very "sparse" presentation which makes the collection process go fast.

## IV.5 Computing a PCGS for permutation groups

If we have a solvable group given already as group of permutations (or other objects, acting on a set in a way to enable use of stabilizer chains, as describes in section II.7) we could obtain a pc-presentation as in section III.10. There is however a more efficient algorithm [Sim90] which we will describe in this section.

The algorithm takes a set of generators $\{a_i\}$ for a permutation group $G$. It returns a PCGS $g_1, \ldots, g_n$ for $G$ that is at the same time also a strong generating set. The corresponding stabilizer chain can be used to determine exponent vectors.

The basic step of the algorithm is that we assume by induction that for a normal subgroup $N \lhd G$ (initializing with the trivial subgroup) we have a PCGS (which is also a strong generating set) and a stabilizer chain. The inductive step then consists of finding elements that together with $N$ will generate a normal subgroup $N \leq M \lhd G$ such that $M/N$ is elementary abelian.

To find such extra generators, consider an element $a \notin N$ such that $a^p \in N$ for a prime $p$. (This second condition can be fulfilled easily by replacing $a$ by a power.) If for all $g \in G$ we have that the commutator $[a, a^g] \in N$ (which implies that $[a^g, a^h] \in N$), then $a$ and its conjugates, together with $N$, generate a normal subgroup $M$ such that $M/N$ is $p$-elementary abelian. Thus $M$ is the subgroup as desired.

Otherwise, there is a conjugate $a^g$ such that $b := [a, a^g] \notin N$. This means that $Nb$ lies in a lower subgroup of the derived series of $G/N$ than $Na$ did. We thus replace $a$ by $b$ and repeat testing the commutators. After each failure the element $Nb$ lies in a lower group of the derived series of $G/N$. If $G$ (and thus $G/N$) is solvable, the last subgroup in this finite series is abelian, which shows that this iteration can happen only a finite number of times. (In fact [Sim90] proposes to use a bound on the derived length, due to [Dix68], to test solvability of an arbitrary group $G$ using the same algorithm.)

Instead of constructing all conjugates $a^g$, we use the idea of algorithm I.21 and form iterated conjugates under generators of $G$, adding only those conjugates $c_i$ that are not yet in the span of the existing elements. A dimension argument shows that the resulting element list $(c_1, c_2, \ldots c_{\dim(M/N)})$ will yield a basis for $M/N$ and that the concatenation of the PCGS for $N$ with $(c_1, c_2, \ldots c_{\dim(M/N)})$ will form a PCGS for $M$.

We also want to modify the new generators to maintain a strong generating set. This is done at the same time as the redundancy test for new candidates for the $c_i$: We start with a stabilizer chain for $N$ and in each step extend this to a stabilizer chain for $\langle N, c_1, c_2, \ldots, c_j\rangle$. If we have this chain and are processing $c_{j+1}$ we sift this element through the chain, using algorithm II.5. If the element sifts through, it is redundant and can be ignored, thus lets assume that the sifting process for $\langle N, c_1, c_2, \ldots, c_j\rangle$ fails on some level $S$ of the stabilizer chain. We then have a partially sifted element $d$ that has been obtained from $c_{j+1}$ by modifying it with elements from $N$ and $c_1, \ldots, c_j$. This means that $\langle N, c_1, c_2, \ldots, c_j, c_{j+1}\rangle = \langle N, c_1, c_2, \ldots, c_j, d\rangle$, and that $d$ could take the place of $c_{j+1}$ in a PCGS. (The changes generating set is simply an IGS with respect to the old generating set.) Furthermore, $|\langle N, c_1, c_2, \ldots, c_j, d\rangle| = p \cdot |\langle N, c_1, c_2, \ldots, c_j\rangle|$ with $p$ prime. Thus if we extend the stabilizer chain layer $S$ by $d$, we get a stabilizer chain for $|\langle N, c_1, c_2, \ldots, c_j, d\rangle|$ and no lower layer in the stabilizer chain needs to be changed. That means that $d$ can be added as part of a strong generating set.

The resulting PCGS/strong generating set thus consists of *sifted* elements obtained from the $c_i$. It obviously refines a normal series of $G$ with elementary abelian steps.

Finally, let us look in more detail at the step of extending the chain at $S$ with the element $d$. This element will normalize $N$ and it stabilizes the base points in the chain above $S$. Thus $d$ normalizes $S$, and as in section II.4 we can extend the orbit in a more efficient way and gives a factorized structure to transversal elements that can be used to improve storage of a factored transversal.

Having determined a PCGS, we would like to determine exponent vectors for elements $x \in G$ with respect to it. Standard permutation group methods will express $x$ as a word in the strong generators, i.e. the PCGS, but not in the required order and with factors possibly repeating. If we only take the generators in the PCGS that are in the topmost elementary abelian layer of the corresponding normal series, the exponent sums for these generators yield the first part of the exponent vector. By dividing off the corresponding product of PCGS elements from $x$ we then fall one layer below and can iterate this process, eventually building the whole exponent vector.

NOTE IV.2: If we are given a permutation group $G$ and a normal subgroup $N \lhd G$ such that $G/N$ is solvable, the same algorithm, initializing with a stabilizer chain for $N$ will produce a chain for $G$ and elements representing a PCGS for $G/N$. This provides an efficient way to represent solvable factor groups as PC groups.

## IV.6 How to do it in GAP

(Such groups are called `PcGroups` in GAP.)
In GAP one can convert a (solvable) permutation group into such a form using the command `IsomorphismPcGroup`.

CHAPTER

# V Rewriting

Rewriting is the formal context in which we use a presentation to bring elements into normal form. To make this deterministic we will consider relations instead of relators and consider them as *rules* from→to.

If you know the basic theory behind Gröbner bases, many ideas will look familiar.

## V.1 Monoids and Rewriting Systems

DEFINITION V.1: A *monoid* is a set with an associative binary operation and an identity element. (In other words: we drop the condition on inverses.)

If $\{x_1, \ldots, x_n\}$ is an alphabet, we consider the set of words (including the empty word) over this alphabet. With concatenation as operation they form a monoid.

We define finitely presented monoids analogous to finitely presented groups. Note however that due to the lack of inverses we have to write in general relations instead of relators.

LEMMA V.2: Every group is a monoid and every finitely presented group is a finitely presented monoid.

Proof: Finitely presented is the only thing that needs showing: If $G = \left\langle \underline{\boldsymbol{g}} \mid R \right\rangle$ we form a monoid generating set $\underline{\boldsymbol{m}} = \{g_1, g_1^{-1}, \ldots\}$ (with $g_i^{-1}$ understood as a formal symbol). Then a monoid presentation is

$$\left\langle \underline{\boldsymbol{m}} \mid \{r = 1 \mid r \in R\} \cup \{g_i g_i^{-1} = 1, g_i^{-1} g_i = 1 \mid 1 \le i \le m\} \right\rangle$$

□

Performance V.3: For finite groups one can often do better, as relations of the form $a^m = 1$ imply the existence of inverses.

We now suppose that we have free monoid $F$ consisting of words in the alphabet $\underline{f}$.

We also assume to have a total ordering $<$ defined on $F$, which fulfills the following conditions:

i) $<$ is a well-ordering: Every nonempty set has a least element. This means that there are no infinite descending sequences.

ii) $<$ is translation invariant: If $a, b, c, d \in F$ and $a < b$ then $cad < cbd$. This implies that the empty word is the smallest element of $F$.

We call such an ordering a *reduction ordering*

Example V.4: Suppose that the elements of the alphabet $\underline{f}$ are totally ordered. Then the "length-plus-lexicographic" ordering on $F$ (i.e. first compare words by length and then lexicographically) is a reduction ordering.

Definition V.5: A *rewriting system* $\mathcal{R}$ on $F$ is a collection of *rules* of the form $a \to b$ with $a, b \in F$ and $b < a$.

If we consider the rules of the rewriting system simply as relations, a rewriting system defines a monoid presentation. Similarly every monoid presentation yields a rewriting system in an obvious way. We will also talk about rewriting systems for groups, meaning the isomorphic monoid.

Given a rewriting system, we consider its rules as methods to "simplify" elements of $F$.

Definition V.6: If $u, v \in F$ we write $u \to v$ (with respect to $\mathcal{R}$) if there is a rule $a \to b$ in $\mathcal{R}$ such that $u$ contains $a$ as a substring (i.e. $u = xay$ with $x, y \in F$ and $v = xby$ is obtained by replacing $a$ in $u$ by $b$.

We write $u \overset{*}{\to} v$ if there is a sequence of words $u_0 = u, u_1, u_2, \ldots, u_{n-1}, u_n = v$ such that $u_i \to u_{i+1}$.

Because the ordering is translation invariant we have that $v < u$ in this case which justifies the idea of "simplification".

Definition V.7: We consider $u, v \in F$ to be equivalent with respect to $\mathcal{R}$, written $u \sim v$, if there is a sequence of words $u_0 = u, u_1, u_2, \ldots, u_{n-1}, u_n = v$ such that $u_i \overset{*}{\to} u_{i+1}$ or $u_{i+1} \overset{*}{\to} u_i$.

It is not hard to see that $\sim$ is in fact the finest equivalence on $F$ defined by $\mathcal{R}$, thus the equivalence classes correspond to the elements of the finitely presented monoid defined by $\mathcal{R}$.

Definition V.8: A word $v \in F$ is called *reduced* if there is no $w \in F$ such that $v \to w$ (with respect to $\mathcal{R}$).

We need the fact that $<$ is a well-ordering to ensure that for $u$ we can compute a reduced element $v$ with $u\overset{*}{\to}v$ in finitely many steps.

ALGORITHM V.9: Given a rewriting system $\mathcal{R}$, and a word $u$, find a reduced word $v$ such that $u\overset{*}{\to}v$.

```
begin
1: $ok := true$;
2: while $ok$ do
3:    $ok := false$;
4:    for Rules $l \to r$ in $\mathcal{R}$ do
5:       if $l$ occurs in $u$ then
6:          Replace $u = alb$ by $arb$;
7:          $ok := true$;
8:       fi;
9:    od;
10: od;
end
```

Note that in general there are multiple ways to apply rules to a word. Thus in general reduced words are not automatically normal forms and $u \sim v$ does not imply that $u\overset{*}{\to}w$ and $v\overset{*}{\to}w$ for a unique element $w \in F$. Our aim is to rectify this.

## V.2 Confluence

We consider three slightly different properties which a rewriting system could have:

DEFINITION V.10: A rewriting system $\mathcal{R}$ on $F$

i) has the *Church-Rosser property* if $u \sim v$ implies that there is $q \in F$ such that $u\overset{*}{\to}q$ and $v\overset{*}{\to}q$.

ii) is *confluent* if $w\overset{*}{\to}u$ and $w\overset{*}{\to}v$ imply that there is $q$ such that $u\overset{*}{\to}q$ and $v\overset{*}{\to}q$.

iii) is *locally confluent* if $w \to u$ and $w \to v$ imply that there is $q$ such that $u\overset{*}{\to}q$ and $v\overset{*}{\to}q$.

LEMMA V.11: Suppose $\mathcal{R}$ has the Church-Rosser Property. Then every $\sim$ class contains a unique reduced element, the canonical representative for this class.

In particular, if we have $u\overset{*}{\to}v$ with $v$ reduced, then $v$ is uniquely determined by $\mathcal{R}$ and $u$.

Proof: Suppose that $u \sim v$ are both reduced. Then by the Church-Rosser property there exists $q$ with $u\overset{*}{\to}q$ and $v\overset{*}{\to}q$. But as $u$ and $v$ are reduced we must have $u = q = v$. □

Corollary V.12: Let $M$ be a monoid given by the rewriting system $\mathcal{R}$. If $\mathcal{R}$ has the Church-Rosser property, then the word problem in this monoid can be solved.

Proof: An element is trivial if its canonical representative is the empty word. □

Testing for the Church-Rosser property seems to be hard. However we will see now that it is in fact equivalent to local confluence, which is much easier to test.

Theorem V.13: For any rewriting system $\mathcal{R}$ with a reduction ordering the Church-Rosser property, confluence and local confluence are equivalent.

Proof: i)$\Rightarrow$ ii): Suppose that $\mathcal{R}$ has the Church-Rosser property and that $w, u, v \in F$ such that $w \overset{*}{\rightarrow} u$ and $w \overset{*}{\rightarrow} v$. Then $u \sim w \sim v$ and thus there exists $q$ such that $u \overset{*}{\rightarrow} q$ and $v \overset{*}{\rightarrow} q$.

ii)$\Rightarrow$ i): Assume that $\mathcal{R}$ is confluent and that $u \sim v$. We want to find a $q$ such that $u \overset{*}{\rightarrow} q$ and $v \overset{*}{\rightarrow} q$.

By definition V.7 we have a sequence of words

$$u_0 = u, u_1, u_2, \ldots, u_{n-1}, u_n = v$$

such that $u_i \overset{*}{\rightarrow} u_{i+1}$ or $u_{i+1} \overset{*}{\rightarrow} u_i$.

We now proceed by induction on $n$. If $n = 0$ we can set $q = u = v$. If $n = 1$ we set $q$ to be the smaller of $u$ and $v$.

Thus assume $n \geq 2$. Then $u_1 \sim v$ and by induction there is $a$ such that $u_1 \overset{*}{\rightarrow} a$ and $v \overset{*}{\rightarrow} a$. If $u_0 \overset{*}{\rightarrow} u_1$, we simply set $q = a$.

If instead $u_1 \overset{*}{\rightarrow} u_0$, by confluence there is $q$ such that $u_0 \overset{*}{\rightarrow} q$ and $a \overset{*}{\rightarrow} q$. But then $v \overset{*}{\rightarrow} q$, as we wanted to show.

ii)$\Rightarrow$ iii): Obvious as local confluence is a special case of confluence.

iii)$\Rightarrow$ ii): Suppose that $\mathcal{R}$ is locally confluent but not confluent. Let $W$ be the set of all words $w$, for which confluence fails. Because $<$ is a well-ordering, there is a smallest element $w \in W$.

Suppose that $w \overset{*}{\rightarrow} u$ and $w \overset{*}{\rightarrow} v$. We want to show that there is $q$ such that $u \overset{*}{\rightarrow} q$ and $v \overset{*}{\rightarrow} q$, contradicting the failure of confluence.

Without loss of generality, we can assume that $u \neq w \neq v$ (otherwise we could set $q = u$ or $q = v$). Consider the first rewriting step of both deductions. We get

$w \to a$ and $w \to b$ with $a\overset{*}{\to}u$ and $b\overset{*}{\to}v$.

$$\begin{tikzcd}
 & & u \arrow[dr] & & \\
 & a \arrow[ur] \arrow[dr] & & d \arrow[dr] & \\
w \arrow[ur] \arrow[dr] & & c \arrow[ur] & & q \\
 & b \arrow[ur] \arrow[dr] & & & \\
 & & v \arrow[uurr] & &
\end{tikzcd}$$

Because we assume local confluence, we know that there is $c$ with $a\overset{*}{\to}c$ and $b\overset{*}{\to}c$.

As $w$ was chosen minimal in $W$, and $a \prec w$, we know that confluence cannot fail at $a$. Thus there is $d$ such that $u\overset{*}{\to}d$ and $c\overset{*}{\to}d$. Therefore also $b\overset{*}{\to}d$.

By the same argument as before, confluence does not fail at $b$. Thus there is $q$ such that $d\overset{*}{\to}q$ and $v\overset{*}{\to}q$. But then $u\overset{*}{\to}q$, which we wanted to show. □

## V.3 The Knuth-Bendix algorithm

As confluent rewriting systems solve the word problem, we would like to obtain such rewriting systems. In this section we will see a method that can be used to modify an existing rewriting system to become confluent. (If you know Gröbner bases, you will find this approach familiar.)

Theorem V.13 tells us that for confluence we only need to ensure local confluence. Suppose that this does not hold, i.e. we have a word $w$ with two reductions $w \to u$ and $w \to v$ but we cannot further rewrite both $u$ and $v$ to the same word $q$.

We want to consider "minimal" failure situations

Lemma V.14: Suppose that local confluence fails at $w$ but not at any proper subword of $w$. Then one of the following holds:

a) $w$ is the left hand side of a rule in $\mathcal{R}$ and contains the left hand side of another rule as subword (probably the whole of $w$).

b) $w = abc$ with nonempty $a, b, c$ and $ab$ and $bc$ both are left hand sides of rules in $\mathcal{R}$.

Proof: Suppose the two reduction are $d \to e$ and $f \to g$. If $d$ and $f$ occur in $w$ without overlap, we can still apply both reductions in either order and obtain the same result $q$:

$$\begin{tikzcd}
q = & a & e & b & g & c \\
v = & a & d \arrow[u] & b & g & c \\
w = & a & d \arrow[d] & b & f \arrow[u] & c \\
u = & a & e & b & f \arrow[d] & c \\
q = & a & e & b & g & c
\end{tikzcd}$$

Thus there needs to be an overlap of $d$ and $f$. This overlap can either have one left hand side completely include the other — that is case a). Otherwise the sides overlap in the form $abc$ with $ab = d$ and $bc = f$. This is case b).

If there is a prefix or suffix to $abc$ in $w$ then $w$ is not minimal. □

Corollary V.15: If local confluence fails at a minimal $w$, then $w = abcd$ such that $b$ is not empty, either $c$ or $d$ is empty and $abc$ and $bd$ are left hand sides of rules.

The basic idea of the method now is that we inspect all such overlaps. If we have a failure of local confluence, i.e. we have $w \overset{*}{\to} u$ and $w \overset{*}{\to} v$ with $u, v$ both reduced and $u \neq v$ we add a new rewriting rule $u \to v$ (or $v \to u$ if $u \prec v$).

This rule will not change the equivalence relation $\sim$, but it will remove this overlap problem.

When continuing on other overlaps, we need of course to consider also overlaps with this new rule. Thus this process may never terminate.

If it terminates (and one can show that it will terminate if there is a finite confluent rewriting system induced by $\mathcal{R}$), we have a confluent rewriting system, which allows us to calculate a normal form for the monoid presented by $\mathcal{R}$.

Algorithm V.16: This method is called (after its proposers) the Knuth-Bendix[1] algorithm.

**Input:** A list $L$ of rules of a rewriting system $\mathcal{R}$.
**Output:** $L$ gets extended by deduced rules so that the rewriting system is confluent.

```
begin
 1: pairs := [];
 2: for $p \in L$ do
 3:    for $q \in L$ do
 4:       Add $(p,q)$ and $(q,p)$ to pairs.
 5:    od;
 6: od;
```

[1]Donald Knuth is also the author of "The Art of Computer Programming" and the creator of TEX. Peter Bendix was a graduate student.

```
7:  while $|pairs| > 0$ do
8:    remove a pair $(p,q)$ from pairs.
9:    for all overlaps $w = xabc$ with left$(p) = ab$ and left$(q) = xbc$ and $(x = \emptyset$
      or $a = \emptyset)$ do
10:     Let $w_p = x \cdot$ right$(p) \cdot c$ and $w_q = a \cdot$ right$(q)$.
11:     Using $L$, reduce $w_p$ to a reduced form $z_p$ and $w_q$ to $z_q$.
12:     if $z_p \neq z_q$ then
13:       if $z_p \prec z_q$ then
14:         Let $r$ be a new rule $z_q \to z_p$.
15:       else
16:         Let $r$ be a new rule $z_p \to z_q$.
17:       fi;
18:       Add $r$ to $L$.
19:       for $p \in L$ do
20:         Add $(p,r)$ and $(r,p)$ to pairs.
21:       od;
22:     fi;
23:   od;
24: od;
end
```

Proof: We are testing explicitly the conditions of lemma V.14 for all pairs. □

NOTE V.17: Analogous to Gröbner bases, one can define a "reduced" confluent rewriting system in which no left hand side can be reduced by the remaining rules.

Analogous to theorem III.24 we remark

THEOREM V.18: If $\mathcal{R}$ is a rewriting system describing the finite group $G$, then the Knuth-Bendix algorithm will terminate after finite time with a confluent rewriting system.

## Arithmetic: Collection

Once we have a confluent rewriting system for a group, we can compute the normal form for any word, and thus compare elements. Typically one would simply assume that all words are reduced. Multiplication of elements then consists of concatenation and subsequent reduction.

In GAP, one can enforce such a reduction for a given finitely presented group with the command `SetReducedMultiplication(G);`

DEFINITION V.19: This process of reduction to normal form is called *collection*[2].

[2]The name stems from the special case of polycyclic presentations for $p$-groups, for which this process has been studied in a purely theoretical context in [Hal33]

Performance V.20: While confluence implies that the order of applying reductions does not have an impact on the final reduced (normal) form, it can have a substantial impact on the runtime. This question has been studied primarily for the case of pc presentations (see below).

## V.4 Rewriting Systems for Extensions

Similar to the situation for presentations (section III.10), we want to construct a rewriting system for a group from rewriting systems for a normal subgroup and for its factor group. Noting that the presentations in III.10 already come with a $G/N$-part and an $N$-part, as well as with conjugacy relations that can be used to write every element as a $G/N$-part and an $N$-part, we simply need to define a suitable order in which these transformations are reducing.

The key to this is to combine two orderings on two alphabets to the so-called wreath product ordering on the union of alphabets:

Definition V.21: Suppose that $<_A$ is an ordering on an alphabet $A$ and $<_B$ and ordering on an alphabet $B$. We consider the disjoint union $A \cup B$. On this set we define the *wreath product ordering* $<_A \wr <_B$ as follows:

We can write a word in $A \cup B$ as a product of words in $A$ and in $B$: Let $v = a_0 b_1 a_1 b_2 a_2 \ldots a_{m-1} b_m a_m$ and $w = c_0 d_1 c_1 d_2 c_2 \ldots c_{n-1} d_n c_n$ with $a_i, c_i \in A^*$ and only $a_1$ or $a_m$ permitted to be the empty word (and ditto for $c$) and $b_i, d_i \in B$ or empty. Then $v <_A \wr <_B w$ if $b_1 b_2 \ldots b_m <_B d_1 d_2 \ldots d_n$, or if $b_1 b_2 \ldots b_m = d_1 d_2 \ldots d_n$ and $[a_0, a_1, \ldots, a_m]$ is smaller than $[c_0, c_1, \ldots, c_n]$ in a lexicographic comparison, based on $<_A$.

Lemma V.22: If $<_A$ and $< B$ are reduction orderings, then $<_A \wr < B$ is.

Note V.23: A rule $ab \to ba'$ with $a, a' \in A^*$ and $b \in B^*$ is reducing with respect to $<_A \wr <_B$. Thus ($A$ representing a normal subgroup and $B$ representing a factor group) wreath product orderings permit to combine rewriting systems of a group from a rewriting system for a factor group and its normal subgroup.

### Complements

If $N \lhd G$ we define as *complement* to $N$ a subgroup $C \leq G$ such that $N \cap C = \langle 1 \rangle$ and $G = NC$. (In other words: $G \cong N \rtimes C$).

Given $G$ and $N \lhd G$ we want to find whether such a complement exists (and later want to consider complements up to conjugacy). Here we will consider the case of $N \cong \mathbb{F}_p^m$ elementary abelian. The case of a solvable $N$ then can be dealt with by lifting methods as described in chapter **??**.

Suppose that $G/N = \langle Ng_1, \ldots Ng_k \rangle$ and that $C$ is a complement to $N$. Then we can find elements $c_i \in C$ such that $C = \langle c_1, \ldots c_k \rangle$ and $Ng_i = Nc_i$. The map $G/N \to C$, $Ng_i \mapsto c_i$ is a homomorphism.

Vice versa every homomorphism $\varphi\colon G/N \to G$, such that $\left\langle N, G/N^{\varphi}\right\rangle = G$ must be (as $|G/N| = |G/N|^{\varphi}$) a monomorphism and define a complement $G/N^{\varphi}$ to $N$ in $G$.

The task of finding a complement therefore is equivalent to the task of finding elements $c_i \in G$ such that the map $G/N \to G$, $Ng_i \mapsto c_i$ is a homomorphism and that $g_i/c_i = n_i \in N$.

We do this by considering the $n_i \in N$ as variables. We want to come up with a system of equations, whose solutions correspond to complements (and unsolvability implies that no complements exist).

For this, suppose that we have a presentation for $G/N$ in the elements $Ng_i$, say $\left\langle \underline{f} \mid r(\underline{f}) = 1, r \in R\right\rangle$. (In practice one would calculate a presentation and then choose the generators accordingly.) We therefore want that

$$1 = r(g_1 n_1, \ldots, g_k n_k). \tag{V.24}$$

We now rewriting these relators with rules reflecting the extension structure: $ng = gn'$ with $n, n' \in N$. As $n' = n^g$ this can be described by the action of $G/N$ on $N$. We also have that $n_i^g n_j^h = n_j^h n_i^g$ because $N$ is abelian.

With these rules we can collect the terms $g_i$ to the left in the same order as in the original relator. Equation (V.24) thus becomes

$$1 = r(g_1, g_2, \ldots, g_k) \prod n_i^{w(\underline{g})}$$

where the $w(\underline{g})$ is a word in the group algebra $\mathbb{F}_p G/N$.

For example, if $r = f_1 f_3 f_2 f_3$, we rewrite as

$$1 = g_1 g_3 g_2 g_3 \cdot n_1^{g_3 g_2 g_3} n_3^{g_2 g_3} n_2^{g_3} g_3 = g_1 g_3 g_2 g_3 \cdot n_1^{g_3 g_2 g_3} n_2^{g_3} g_3^{g_2 g_3 + 1}$$

In this expression we can explicitly evaluate $r(g_1, \ldots, g_n) \in N$ (which will give the (inverse of) the right hand side of linear equations. If we consider each $n_i = (n_{i,1}, \ldots, n_{i,m})$ as a column vector with variable entries, the remaining part $\prod n_i^{w(\underline{g})}$ yields linear equations in the variables $n_{i,j}$.

Considering all relators thus gives an inhomogeneous system of equations, whose solutions describe complements.

Example V.25: Consider $G = S_4 = \langle a = (1,2,3,4), b = (1,2)\rangle$ and the normal subgroup $N = \langle (1,2)(3,4), (1,3)(2,4)\rangle \lhd G$. Then $a \mapsto (1,3)$, $b \mapsto (1,2)$ is a homomorphism with kernel $N$. The action of $G$ on $N$ is described by the matrices $a \mapsto \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and $b \mapsto \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$.

The factor group $G/N \cong S_3$ has (in these images) the presentation

$$\left\langle x, y \mid x^2 = y^2 = (xy)^3 = 1\right\rangle.$$

We now want to find elements of the form $an$, $bm$ (with $n, m \in N$) which fulfill these relations. We get the following equations:

$$\begin{aligned}
x^2 : (an)^2 &= a^2 n^a n = a^2 n^{a+1} \\
y^2 : (bm)^2 &= b^2 m^b m = b^2 m^{b+1} \\
(xy)^3 : (anbm)^3 &= (ab)^3 n^{babab} m^{abab} n^{bab} m^{ab} n^b m = (ab)^3 n^{babab+bab+b} m^{abab+ab+1}
\end{aligned}$$

We now assume $N$ as a 2-dimensional vector space over $\mathbb{F}_2$ with basis as given. Thus $n = [n_1, n_2]$ and $m = [m_1, m_2]$. We also evaluate the expressions $a^2 = (1,3)(2,4) = [0,1]$, $b^2 = () = [0,0]$, $(ab)^3 = () = [0,0]$. The equations thus become in vector form:

$$\begin{aligned}
-a^2 = [0,1] &= n^{a+1} = [n_1, n_2] \cdot \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = [0, n_1] \\
-b^2 = [0,0] &= m^{b+1} = [m_1, m_2] \cdot \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = [m_2, 0] \\
-(ab)^3 = [0,0] &= n^{babab+bab+b} m^{abab+ab+1} \\
&= [n_1, n_2] \cdot \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + [m_1, m_2] \cdot \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = [0,0]
\end{aligned}$$

which yields the following system of (nontrivial) equations:

$$\begin{aligned}
n_1 \qquad &= 1 \\
m_2 &= 0
\end{aligned}$$

whose solutions correspond to complements. For example the solution $n_1 = m_1 = 1$, $n_2 = m_2 = 0$ corresponds to the generators

$$(1,2,3,4) \cdot (1,2)(3,4) = (2,4) \qquad \text{and} \qquad (1,2) \cdot (1,2)(3,4) = (3,4).$$

Note V.26: As classes of complements are parameterized by the 1-Cohomology group this process can also be considered as a way of calculating 1-Cohomology groups.

## Polycyclic Presentations

Let us now return to the pc presentations for solvable groups, which we studied in section IV.1:

We have power relations $g_i^{p_i} = v_i(g_{i+1}, \ldots, g_n)$. We consider the conjugation rules (for $j > i$) of the form $g_j^{g_i} = w_{i,j}(g_j, \ldots, g_n)$ as rewriting rules

$$g_j g_i = g_i w_{i,j}(g_j, \ldots, g_n)$$

with respect to an (iterated) wreath product ordering.

The confluence condition V.14 yields the following easy consequence

Corollary V.27 ([Wam74][3]): A presentation of a form as given in section IV.1 with $2 \leq p_i < \infty$ for every $i$ yields a confluent rewriting system, if the following conditions hold:

$$\begin{array}{rrcll}
\text{Overlap} & \text{Reduction 1} & & \text{Reduction2} & \\
g_i^{p_i+1} = & g_i v_i & = & v_i g_i & \\
g_j^{p_j} g_i = & v_j g_i & = & g_j^{p_j-1} g_i w_{i,j} & (i<j) \\
g_j g_i^{p_i} = & g_j v_i & = & g_i w_{i,j} g_i^{p_i-1} & (i<j) \\
g_k g_j g_i = & g_j w_{j,k} g_i & = & g_k g_i w_{i,j} & (i<j<k)
\end{array}$$

(There are generalizations for infinite polycyclic groups.)

Proof: These are all possible overlaps. □

Performance V.28: One can show that a subset of such conditions suffices.

This test makes it possible to test whether an arbitrary presentation that syntactically looks like a PC presentation really is one. We will use this as a tool for finding quotients of given finitely presented groups, essentially trying to write down a pc presentation that is compatible with being the quotient of a given group.

## V.5 Constructing extensions, 2-cohomology

## V.6 Quotient Algorithms

As an application of extensions, we consider again the problem of finding quotients of a finitely presented group.

Definition V.29: A *variety* of groups is a class of groups that is closed under subgroups, factor groups and direct products

Examples of varieties are solvable groups, nilpotent groups, $p$-groups or abelian groups. Furthermore one could impose for example conditions on the length of certain "natural" normal series.

Lemma V.30: Let $G$ be a group. For every variety $\mathcal{V}$ there is a smallest normal subgroup $N_{\mathcal{V}} \lhd G$ such that $G/N_{\mathcal{V}}$ is in $\mathcal{V}$.

Proof: If $N, M \lhd G$ both have the property, then $G/(N \cap M)$ is a subdirect product of $G/N$ with $G/M$. □

The quotient algorithms we will study aim to construct for a finitely presented group $G$ the largest quotient group $F = G/N$ in a certain variety, possibly subject to conditions of order or length of a composition series. (The lemma shows that this is a sensible aim to have.)

[3]Originally proven directly for nilpotent groups without recuse to rewriting systems

Note V.31: We could apply such an algorithm in particular to free groups. This gives an – initially crude – way of constructing all groups of a given order in a particular variety. The difficulty however is the elimination of isomorphic groups.

This approach has been refined for particular cases, incorporating a reasonably efficient rejection of isomorphic duplicates. The resulting algorithms are the $p$-group generation algorithm [O'B90] and the "Frattini extension" algorithm to construct all solvable groups of a given order [BE99].

The idea behind the quotient algorithms is as follows:

Assume that we know already a homomorphism $\varphi\colon G \to H$ which represents a smaller quotient (often the largest quotient in which the length of a normal series is bounded by one less, than we want to acceive).

We now want to find a larger quotient $\lambda\colon G \to E$ such that $\operatorname{Kern}\lambda < \operatorname{Kern}\varphi$ and $M = \operatorname{Kern}\varphi / \operatorname{Kern}\lambda$ is elementary abelian. (Iteration then allows to deal with arbitrary solvable $M$. Similar to the lifting paradigm **??**, a solvable $M$ really is the relevant case.) We thus have that $E$ must be an extension of $M$ by $H$. We temporarily ignore the (obvious) question of what $M$ actually is, but assume that we already have $M$ – we will study this specifically for the different algorithms.

To describe $E$ we want to use a rewriting system. We assume that we have already a (confluent) rewriting system for $H$ (as $M$ is elementary abelian, we also know a rewriting system for it) and build a rewriting system for $E$ using the wreath product ordering.

In this new rewriting system every rule $l \to r$ for $H$ now becomes $l \to r \cdot m$, with $m \in M$ in $E = M.H$. We call $m$ a *tail* for the rule. Because we don't know the correct values for $m$, we consider these tails as variables. We also need rules that describe the action of $H$ on $M$.

Since we assumed the rewriting system for $H$ to be confluent, any overlap of left hand sides of rules in $H$ reduces uniquely. If we consider the *same* overlap of the corresponding (extended) rules in $E$, we get potentially two reductions, products of the form $hm_1$ and $hm_2$, and thus the condition that $m_1 = m_2$. (The confluence in $H$ implies that the $H$-part in both reductions must be the same.) Here both $m_1$ and $m_2$ are products of conjugates of tails. The confluence condition for $E$ thus yields (linear) equations in the tails, very similar to the situation of complements in section V.4.

These equations need to be satisfied for $E$ to be a proper group. A second set of equations comes from the fact that $E$ should be a quotient of $G$. The homomorphism $\varphi$ gives us the images of the generators of $G$ in $E$. The images of these generators in $E$ must have the same $H$-part (but again might have to be modified with a variable $M$-tail). We now evaluate the relators of $G$ in these images. Again, because $\varphi$ is a homomorphism, we get equations involving only the tails.

We will now use these equations not only to describe the proper extension, but also to actually determine the largest possible $M$. The details of this differ, depending on the kind of quotients we are searching for.

## $p$-Quotients

Let us first consider the case of the quotients being $p$-groups (finite nilpotent groups, being the direct product of $p$-groups are essentially equivalent).

We need a bit of theory about generating systems of $p$-groups:

Definition V.32: Let $H$ be a finite group. The *Frattini-subgroup* $\Phi(H) \leq H$ is the intersection of all maximal subgroups of $H$.

Lemma V.33: $\Phi(H)$ consists of those elements that are redundant in every generating set of $H$.

Proof: Homework. □

Theorem V.34 (Burnside basis theorem): Let $H$ be a finite $p$-group. Then $\Phi(H) = H'H^p$. If $\left[H{:}\Phi(H)\right] = p^r$, every set of generators of $H$ has a subset of $r$ elements, which also generates $H$.

Proof: Suppose $M \leq H$ maximal. Then (as $p$-groups are nilpotent $N_H(M)$ is strictly larger than $M$) we have that $M \lhd H$ and $\left[H{:}M\right] = p$ and thus $H'H^p \leq M$.

On the other hand $H/H'H^p$ is elementary abelian, and thus $\Phi(H/H'H^p) = \langle 1\rangle$. But this implies that $\Phi(H) \leq H'H^p$.

Finally let $\underline{\boldsymbol{h}} = \{h_1, \dots, h_n\}$ be a generating set for $H$. Then $\{\Phi(H)h_i\}_{i=1}^n$ must generate $H/\Phi(H)$, which is an $r$-dimensional vector space. Thus we can find a subset $B = \{h_{i_1}, \dots, h_{i_r}\} \subset \underline{\boldsymbol{h}}$ such that $\{\Phi(H)h_{i_1}, \dots, \Phi(H)h_{i_r}\}$ is a basis of $H/\Phi(H)$. But then $H = \langle B, \Phi(H)\rangle = \langle B\rangle$. □

We note, that $H'H^p = \Phi(H)$ is the first step of the lower $p$-elementary central series of $H$. (This series is defined by $L_i = [H, L_{i-1}]L_{i-1}^p$ and has the property that $L_{i-1}/L_i$ is $p$-elementary abelian and central in $H/L_i$.)

Our aim in constructing $p$-group quotients of a finitely presented group $G$ now will be to construct these quotients in steps along this series. the first step therefore will be $G/G'G^p$. (We can determine this subgroup using the abelian quotient, section III.9, by simply imposing further relations $x^p = 1$ on all generators $x$.) By rearranging the generators of $G$, and by choosing the basis for $G/G'G^p$ suitably, we will assume that the images of $g_1, \dots, g_k$ form a basis for $G/G'G^p$.

All further steps now fit the general scheme described in the previous section. We assume the existence of a homomorphism $\varphi\colon G \to H$ onto a $p$-group $H$ such that $\operatorname{Kern}\varphi \leq G'G^p$. We want to find a larger quotient $\lambda\colon G \to E$ with $M := (\operatorname{Kern}\varphi)^\lambda \lhd E$ being $p$-elementary abelian. Because of the choice of series to construct we can assume that

- $M$ is central in $E$ – i.e. the relations for the action of $H$ on $M$ are trivial.

- $M \leq \Phi(E)$, i.e. we do not need to consider any extra generators for $M$. $M$ is simply generated by all the tails, and the solution space to the equations in the tails is the largest possible $M$.

We also assume that we have a pc presentation for $H = \langle \underline{\boldsymbol{h}} \rangle$ in the generating set $\underline{\boldsymbol{h}} = \{h_1, \ldots, h_n\}$. We shall assume that $\langle h_{k+1}, \ldots, h_n \rangle = \Phi(H)$, i.e. the (images of) the generators $h_i, \ldots, h_k$ generate $H/\Phi(H)$ and (by theorem V.34) thus $H = \langle h_1, \ldots, h_k \rangle$. As a consequence of this, we can write for $i > k$ every $h_i$ as a product of $h_1, \ldots, h_k$. This product expression is implicitly maintained by the assumption that for every $i > k$ there is a relation in the pc presentation for $H$ in which $h_i$ occurs only once and with exponent one, and all other occuring generators have index $< i$. We call this relation the *definition* of $h_i$.

This condition is trivial in the first step, it will be maintained for all new generators introduced in the process described here.

Now assume that $\varphi$ is given by the images $g_i^\varphi$ for the generators of $G$ in $H$. By the choice of basis for $G/G'G^p$ we can assume that $g_i^\varphi = h_i$ for $i \leq k$ (basically this is the definition of $h_i$ for $i \leq k$). The further generator images are expressed as words $g_i^\varphi = v_i(\underline{\boldsymbol{h}})$ in the generators of $H$.

We now want to construct the larger group $C$ which is an extension of $M$ by $H$ by forming a rewriting system for $C$ as described. (Eventually we shall want a quotient $E$ of $G$, this group $E$ will be obtained as a quotient of the group $C$ which we shall construct first.)

Since elements of $H$ become cosets in $C$ there is potentially a choice which representatives for elements of $H$ to choose in $C$. We settle this choce with the convention that we maintain the definitions of the previous level, i.e. for $i \leq k$ we have that $h_i$ shall be the image of $g_i$, and that for $i > k$ the generator representing $h_i$ shall be defined as a word in $h_i, \ldots, h_{i-1}$ using the definition relation for $h_i$ in $H$.

This convention implies, that a relation $l_j(\underline{\boldsymbol{h}}) \to r_j(\underline{\boldsymbol{h}})$ of $H$ which is a definition simply is maintained in $C$, if it is not a definition, it gets modified to $l_j(\underline{\boldsymbol{h}}) \to r_j(\underline{\boldsymbol{h}}) \cdot m_j$, where the $m_j$ are variables, representing the elements of $M$.

We also (implicitly, by incorporating this in the collection process) add relations $m_j^p = 1$ and – reflecting the planned centrality of $M$ – relations $m_j h_i = h_i m_j$.

Next we check the confluence relations V.27, using the fact that the rewriting system for $H$ is in fact a pc presentation, for all relations obtained from $H$ in this way. Since the relations for $H$ are confluent this yields (linear, homogeneous) equations in the $m_i$. We now let $M$ be the solution space for this system of equations. It will have a basis consisting of some of the $m_i$, while other $m_j$ are expressed as products.

Because the elements of this $M$ fulfill the equations, the modified relations describe an extension $C = M.H$ of $M$ by $H$. It is the largest extension of this type such that $M$ is elementary abelian, central and contained in $\Phi(C)$. This group $C$ is called the *p-covering group* of $H$.

Each of the $m_i$ surviving in a basis of $M$ arose as tail in a modified relation. This new relation for $C$ is considered as the definition of $m_i$ in the next iteration. If $m_j$ was not chosen as basis element the relation $l_j(\underline{\boldsymbol{h}}) \to r_j(\underline{\boldsymbol{h}}) \cdot m_j$ is not a definition, and $m_j$ is replaced by the appropriate product of the basis elements.

The construction so far only involved $H$. In a second step we now want to go from the covering group $C$ to a – possibly smaller – quotient $E$, which also is a quotient of $G$, together with the map $\lambda\colon G \to E$. To do this, we need to determine the images of the generators of $G$ under $\lambda$ and evaluate the relations for $G$ in these images.

We shall assume still that $g_i^\lambda = h_i$ for $i \le k$. (As $C = \langle h_1, \ldots, h_k \rangle$ this automatically guarantees that $\lambda$ is surjective.) For all further generators of $g$ we know that $g_i^\varphi = w_i(\underline{\boldsymbol{h}})$ for some word $w_i$. This relation holds in $H$, but in the larger group $E$ might need to be modified by an element of $M$. We thus set $g_i^\lambda = w_i(\underline{\boldsymbol{h}}) \cdot l_i$ with $l_i \in M$ another variable (which will be solved for).

Then for each relator $r(\underline{\boldsymbol{g}})$ of $G$, we evaluate $r(\{g_i^\lambda\})$. These evaluations yield elements of $M$ (as the relations hold in $C/M = H$), expressed as words in the $m_i$ and the $l_j$. We use these equations to determine values for the $l_j$ in terms of the $m_i$, also they define further relations amongst the $m_i$. If we consider the subgroup $M_1$ generated by these relations, the factor $M/M_1$ is the largest central step consistent with a lift $\lambda$, thus $E := C/M_1$ is the quotient we were looking for. (This will eliminate some variables $m_i$ from a basis, the corresponding relations are not considered definitions any more.)

We finally notice that we now have a larger quotient $E$, an epimorphism $\lambda\colon G \to E$ and a designation of relations as definitions as needed for the next iteration. The process stops if either at some step $M/M_1$ is trivial (and thus $E = H$), or a pre-set maximal length or order of the quotient is reached.

## An Example

To illustrate this process, consider the (infinite) group $G = \left\langle x, y | x^3 = y^3 = (xy)^3 = 1 \right\rangle$, we are searching for 3-quotients.

The largest abelian quotient is easily seen to be $\left\langle a, b \mid a^3 = b^3 = 1, ba = ab \right\rangle$ with an epimorphism $x \mapsto a$, $y \mapsto b$.

For the first lifting step (no relation is a definition so far), we introduce variables $c, d, e$ as tails, and get relations $a^3 \to c$, $b^3 \to d$, $ba \to abe$. (We will implicitly assume that $\langle c, d, e \rangle$ is central of exponent 3.) According to V.27, we now need to consider the following overlaps, none of which are new:

| Overlap | Reduction 1 | Reduction 2 | Equation |
|---|---|---|---|
| $a \cdot a^2 \cdot a$ | $ca$ | $ac$ | $ca = ac$ |
| $b \cdot b^2 \cdot b$ | $db$ | $bd$ | $db = bd$ |
| $b \cdot a \cdot a^2$ | $abea^2 \to a^3be^3 \to bce^3$ | $bc$ | $e^3 = 1$ |
| $b^2 \cdot b \cdot a$ | $da \to ad$ | $b^2abe \to ab^3e^3 \to ade^3$ | $e^3 = 1$ |

We get the 3-covering group

$$\begin{aligned}\langle a,b,c,d,e \quad | \quad & a^3 \to c, b^3 \to d, ba \to abe, c^3, d^3, e^3,\\ & [a,c],[a,d],[a,e],[b,c],[b,d],[b,e],[c,d],[c,e],[d,e]\rangle\end{aligned}$$

of order $3^5 = 243$.

Now we impose the condition to be a quotient of $G$. With the inherited setting $x \to a$, $y \to b$ (as $G$ has only two generators, no variables $l_i$ have to be introduced), the relators become

| Relator | Evaluation |
|---|---|
| $x^3$ | $1 = a^3 \to c$ |
| $y^3$ | $1 = b^3 \to d$ |
| $(xy)^3$ | $1 = (ab)^3 \to a^2bebab \to a^3b^3e^3 \to cde^3$ |

from which we conclude that $c = d = 1$ and $e^3 = 1$. At the end of this step, the quotient is

$$\langle a,b,e \mid a^3 \to 1, b^3 \to 1, ba \to abe, e^3, ea \to ae, eb \to be\rangle$$

which is the nonabelian group of order 27 and exponent 3. The relation $ba \to abe$ is the definition of $e$.

In the next iteration, we append tails to all non-definition relations and get the relations

$$a^3 \to c, b^3 \to d, ba \to abe, e^3 \to f, ea \to aeg, eb \to beh,$$

together with the implicit condition that $\langle c,d,f,g,h\rangle$ is central of exponent 3. (Here we introduced tails $c$ and $d$ anew, as above, hoever the relations for $G$ will impose that both must be trivial. We therefore simplify already at this point to $c = 1$ and $d = 1$ to reduce the example size.) Note that $ba \to abe$ was a definition, and therefore got no tail.

Since we set $c = d = 1$ the overlaps of $a^3$ and $b^3$ with itself are not of interest. The overlap $b \cdot a \cdot a^2$ now yields

$$b = b \cdot a \cdot a^2 \to abea^2 \to abaega \to aba^2eg^2 \to a^2beaeg^2 \to a^3be^3g^3 \to bf$$

and thus $f = 1$. (Similarly, it also follows from the overlap $b^2 \cdot b \cdot a$.)

With this reduction, all other overlaps yield no new relations:

| Overlap | Reduction 1 | Reduction 2 | Equation |
|---|---|---|---|
| $e \cdot b \cdot a$ | $beha \to baegh \to abe^2gh$ | $eabe \to aebeg \to abe^2gh$ | |
| $e \cdot e^2 \cdot e$ | $e$ | $e$ | |
| $e \cdot a \cdot a^2$ | $aega^2 \to a^3eg^3 \to eg^3$ | $e$ | $g^3 = 1$ |
| $e \cdot b \cdot b^2$ | $behb^2 \to eh^3$ | $e$ | $h^3 = 1$ |
| $e^2 \cdot e \cdot a$ | $a$ | $e^2aeg \to eae^2g^2 \to ag^3$ | $g^3 = 1$ |
| $e^2 \cdot e \cdot b$ | $b$ | $e^2beh \to ebe^2h^2 \to bh^3$ | $h^3 = 1$ |

Evaluating the relators, the only interesting image is the image of $(xy)^3$ which yields

$$1 = (ab)^3 \to a^2beabeb \to a^2baegb^2eh \to a^3b^2e^2begh^3 \to b^3e^3gh^2 \to gh^2$$

which — together with the relation $h^3 = 1$ implies that $g = h$. The next quotient thus is the following group of order $3^4 = 81$:

$$\langle a, b, e, g \mid a^3 \to 1, b^3 \to 1, ba \to abe, e^3, ea \to aeg, eb \to beg, [g,a], [g,b], [g,e] \rangle$$

where $ea \to aeg$ is the definition of $g$.

## Solvable Quotients: Lifting by a module

We now want to generalize this process to a larger class of quotients. However, as soon as we wantto consider not only $p$-groups (or finite nilpotent groups), a couple of problems arise:

- Which primes do we need to consider for $M$?
- $M$ is not any longer central, we need to consider an action of $H$ on $M$. How do we represent conjugates?
- We cannot assume any longer that $M \leq \Phi(E)$ – so $M$ might not be generated solely by the tails of rules, nor is a lift of the previous homomorphism automatically surjective.

For solvable groups, these problems have been addressed by two different approaches, producing "Solvable Quotient" algorithms:

The approach of [Ple87] constructs for the relevant primes $p$ all irreducible $H$ modules over the field with $p$ elements. This construction is done using the composition structure of $H$. For each module $M$ (i.e. for a defined action of $H$), the algorithm then constructs all extensions $M.H$ and tests, whether the given homomorphisms $\varphi$ can be lifted to any of these extensions.

The second approach instead tries to determine $M$ from relations. To deal with the issue of conjugation, it introduces not ony tail variables $m_i$, but also $H$-conjugates

$m_i^h$. Rewriting thus does not any longer produce a system of linear equations, but a *module presentation* for $M$. A process, similar to Coset enumeration, called *module enumeration* [Lin91] then is used to determine a basis for $M$ as well as matrices for the action of $H$ on $M$.

## Hybrid Quotients

Nothing in the methods for solvable groups (with the possible exception of the method for identifying primes) really requires $H$ to be solvable, but many calculations become much harder, making a direct generalization infeasible.

Instead, a better approach [DH21] generalizes the concept of the $p$-covering group. We describe the facts, but refer to the paper for proofs:

We consider, as before, the situation of a given finite quotient $\varphi\colon G \to H$ and a chosen prime $p$. We want to find a quotient $\lambda\colon G \to E$ such that $\operatorname{Kern}\lambda \le \operatorname{Kern}\varphi$ and that $M = (\operatorname{Kern}\varphi)^\lambda$ is an elementary abelian $p$-group. Since we can iterate such a process, we can assume, without loss of generality, that $M$ is minimally normal. In fact we will make a slightly weaker assumption, namely that $M$ is a direct product of minimal normal subgroups of $E$, which are mutually isomorphic in a way that is compatible with the conjugation action that is induced by elements of $H$. (Or, in other language, the case that $M$ is a direct sum of simple, isomorphic, $\mathbb{F}_pH$ modules.) We call such an $M$ *homogeneous*.

We thus now consider the case of a simple $\mathbb{F}_pH$ module $V$. For ease of argument,assume that $V$ remains simple over the algebraic closure of $\mathbb{F}_p$. We use $e$ to denote the number of generators of $G$. Based on Gaschütz [Gas54], the work in [DH21] now shows:

- There is a finite group $\hat{H}_{V,e}$, generated by $e$ elements such that

  1. $\hat{H}_{V,e}$ has an elementary abelian normal subgroup $L \lhd \hat{H}_{V,e}$ with $\hat{H}_{V,e}/L \cong H$.
  2. $L$ is the direct product of minimal normal subgroups, all isomorphic to $V$ as $\mathbb{F}_pH$ modules.
  3. *Any* finite, $e$-generated, group with properties i) and ii) is isomorphic to a quotient of $\hat{H}_{V,e}$.

- $\hat{H}_{V,e}$ is a subdirect product of:

  1. $e$ groups, explicitly described as subgroups of a semidirect product $V^{\dim V} \rtimes H$.
  2. Extensions of $H$ by $V$, corresponding to a basis of the 2-cohomology group $H^2(H, V)$. (Furthermore, one can show that this cohomology group must be trivial, unless $V$ lies in the principal $p$-block for $H$.)

This group $\hat{H}_{V,e}$ is called the universal $(V,e)$-cover.

If $H$ is a $p$-group, there will be only one choice of $V$ (namely the 1-dimensional trivial module) and $\hat{H}_{V,e}$ will be equal to the $p$-covering group defined above.

The algorithm in **??**, called *hybrid quotient* algorithm, now constructs this cover explicitly. As in the $p$-Quotient algorithm, the quotient $\varphi G \to H$ is extended, via the underlying free group $F$ of $G$, to a homomorphism $\psi\colon F \to \hat{H}_{V,e}$ by mapping a generator $f$ of $F$ to an element[4] of $\hat{H}_{V,e}$, whose image in $H$ is that of $f$ under $\varphi$.

Evaluating the relators defining $G$ under $\psi$ then yields a normal subgroup $N \lhd \hat{H}_{V,e}$ such that $\hat{H}_{V,e}/N$ is the maximal quotient with a kernel that is $V$-homogeneous.

## V.7 How to do it in GAP

### Quotient Algorithms

`EpimorphismPGroup(`$G$`,`$p$`,`$c$`)` implements the $p$-quotient algorithm for finding a quotient group $P$ of $p$-class (length of the lower $p$-central series) $c$, if $c$ is ommitted the largest $p$-quotient $P$ is calculated. The function returns a homomorphism $G \to P$.

A solvable quotient algorithm is provided by `EpimorphismSolvableGroup(`$G$`,`$n$`)`, which tries to find a quotient of order up to $n$ (using only primes involved in $n$); respectively by `EpimorphismSolvableGroup(`$G$`,`*primes*`)`, which finds the largest solvable quotient whose order involves only the given primes.
A hybrid quotient algorithm is available under `https://github.com/hulpke/hybrid/hybrid.g`

[4]one shows, that the choice of it is not important

# Bibliography

[Art55] Emil Artin, *The orders of the classical simple groups*, Comm. Pure Appl. Math. **8** (1955), 455–472.

[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.

[Cam81] Peter J. Cameron, *Finite permutation groups and finite simple groups*, Bull. London Math. Soc. **13** (1981), 1–22.

[Cam99] ———, *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.

[CCN$^+$85] J. H. Conway, R. T. Curtis, S. P. Norton, R. A. Parker, and R. 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.

[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.

[Deh11] Max Dehn, *Über unendliche diskontinuierliche Gruppen*, Math. Ann. **71** (1911), 116–144.

[DH21] Heiko Dietrich and Alexander Hulpke, *Universal covers of finite groups*, J. Algebra **569** (2021), 681–712.

[Dix68] John D. Dixon, *The solvable length of a solvable linear group*, Math. Z. **107** (1968), 151–158.

[Dix69] ______, *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*, in Kantor and Seress [KS01], pp. 155–168.

[Eic01] Bettina Eick, *Computing with infinite polycyclic groups*, in Kantor and Seress [KS01], pp. 139–154.

[Fir05] David Firth, *An algorithm to find normal subgroups of a finitely presented group, up to a given finite index*, Ph.D. thesis, Warwick University, 2005.

[Gas54] Wolfgang Gaschütz, *Über modulare Darstellungen endlicher Gruppen, die von freien Gruppen induziert werden*, Math. Z. **60** (1954), 274–286.

[Gor82] Daniel Gorenstein, *Finite simple groups*, Plenum Press, 1982.

[GP06] Alexander Gamburd and Igor Pak, *Expansion of product replacement graphs*, Combinatorica **26** (2006), no. 4, 411–429.

[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.

[HS01] Alexander Hulpke and Ákos Seress, *Short presentations for three-dimensional unitary groups*, J. Algebra **245** (2001), 719–729.

[Hul05] Alexander Hulpke, *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] Daniel Kunkle and Gene Cooperman, *Twenty-six moves suffice for Rubik's cube*, ISSAC 2007, ACM, New York, 2007, pp. 235–242.

[KS01] William M. Kantor and Ákos Seress (eds.), *Groups and computation III*, Ohio State University Mathematical Research Institute Publications, vol. 8, Berlin, de Gruyter, 2001.

[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.

[Lin91] S. A. Linton, *Constructing matrix representations of finitely presented groups*, J. Symbolic Comput. **12** (1991), no. 4-5, 427–438.

[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.

[LS97] Eugene M. Luks and Ákos Seress, *Computing the fitting subgroup and solvable radical for small-base permutation groups in nearly linear time*, Proceedings of the 2nd DIMACS Workshop held at Rutgers University, New Brunswick, NJ, June 7–10, 1995 (Providence, RI) (Larry Finkelstein and William M. Kantor, eds.), DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, vol. 28, American Mathematical Society, 1997, pp. 169–181.

[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.

[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.

[Nov55] P. S. Novikov, *Ob algoritmičeskoĭ nerazrešimosti problemy toždestva 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. 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.

[Pra90] Cheryl E. Praeger, *The inclusion problem for finite primitive permutation groups*, Proc. London Math. Soc. (3) **60** (1990), no. 1, 68–88.

[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.

[Rok10] Tomas Rokicki, *Twenty-two moves suffice for rubik's cube*, The Mathematical Intelligencer **32** (2010), 33–40.

[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.