Alexander Hulpke

Some Examples of Computer Calculations with Finitely Presented Groups

Zusammenfassung: We illustrate, in three examples taken from joint work with Gerhard Rosenberger, how standard computer algorithms can be used to verify finitely presented groups being finite, infinite, or having a free subgroup.

Schlagwörter: Finitely Presented Group, Computer, Calculation

Klassifikation: 20-04, 20F05

For Gerhard Rosenberger,
who has been a Guide to New and Exciting Mathematical Lands.

When investigating a finitely presented group G, the most fundamental questions are whether G is finite, and if so to determine a faithful representation); respectively for infinite groups to find structural information (such as free subgroups) that prove infinity. In this quest computer calculations can be of help, even if their success cannot be guaranteed a priori.

Such methods have been long known, some predate the existence of electronic computers. In the context of modern systems for computational group theory (such as GAP [GAP], which we shall use below, or Magma [BCP97]), however, they can also be used in interplay with explicit calculations in finite quotients. This allows for more systematic search for subgroups and homomorphisms, and ultimately gives a far broader scope to many of the algorithms.

The purpose of this note is to describe the algorithms available, and illustrate their use through examples, taken from joint work with Gerhard Rosenberger. In this, our goal is not to provide explanations of details, or correctness proofs, for the by now classical algorithms. This has been done before [Neu82, Sim94, HEO05], much better than the author would be able to do here. Instead, the goal is to describe how to use the existing tools to obtain the desired mathematical results.

As guidance for the user we also indicate a scope for these methods – roughly what kind of calculation one can expect to finish, on a common laptop in under half an hour and using under 2GB of memory, for a 2-generator finitely generated

Alexander Hulpke, Department of Mathematics, Colorado State University, 1874 Campus Delivery, Fort Collins, CO, 80523-1874, e-mail: hulpke@colostate.edu

https://doi.org/ENTER DOI HERE

group. With the cost of many of the algorithms being exponential, this will not change dramatically, even if more resources are available.

The algorithms we shall encounter fall, roughly, in two categories. One finds quotients (and works in such quotients), the other manipulates words in generators. For solving concrete problems, usually a mix of both approaches is required.

1 Impossibility Results

Before looking at actual algorithms, it is appropriate to address the resident elephant in the room. This is the result by Boone [Boo57] and Novikov [Nov55] that there cannot be a universal algorithm that determines whether a given finite presentation describes the trivial group. This result is proven by reduction to the Halteproblem for Turing machines. As such, it applies to any "classical" computer, as well as to quantum computers.

While it is a genuine impossibility result, its interpretation can be misleading at first. The result says: "*you are not guaranteed to win*". But this does not mean: "*you must lose*". Indeed, we will see algorithms[1] that produce (exact, proven) results in many cases. The impossibility result manifests itself in that an algorithm might end in failure, or might not terminate within available predefined memory or time constraints. In the case of such failures, we cannot determine anything about the problem considered – maybe we were just not patient enough to wait or had insufficient memory available.

2 Fantastic homomorphisms and how to find them

Let $G = \langle \underline{\mathbf{g}} \rangle$ be a group. By the universal property of free products (this is often called van Dyck's theorem), a map $\varphi\colon \underline{\mathbf{g}} \to H$ to a group H, given on generators, extends to a homomorphism $G \to H$ if and only if for each relator $r \in R$ the evaluated image $r(\varphi(\underline{\mathbf{g}})) = 1$ is trivial. Thus, as long as H is a group in which element equality can be tested easily, one can test whether such maps are homomorphisms. Evaluating them on arbitrary elements of G is easy.

1 We shall use the word "algorithm", rather than "method", despite the fact that we cannot promise successful completion a priori.

2.1 Working with homomorphisms

Assuming that H is a group for which effective algorithms exists, for example a permutation group, or a so-called PC-group (a group with a normal form given by a polycyclic presentation, [LNS84]), one then can work in the image $\varphi(G) \leq H$ to find suitable subgroups or elements.

Such homomorphic images can also be used to represent subgroups [Hul01]: The subgroup $U \leq G$ is represented, using a suitable homomorphism φ with $\ker\varphi \leq U$, by the pair $(\varphi, \varphi(U) \leq \varphi(G))$. For any subgroup represented this way, we denote by `DefiningQuotientHomomorphism` the homomorphism φ used to represent it.

In this representation we can test membership in U (as $\varphi(x) \in \varphi(U)$ for $x \in G$) or the subgroup index $[G : U] = [\varphi(G) : \varphi(U)]$. The normalizer $N_G(U)$ is represented by the pair $[\varphi, N_{\varphi(G)}(\varphi(U))])$. For the intersection of two subgroups, we form the direct product of their respective homomorphisms and calculate the intersection in its image.

With this reliance on homomorphisms, the basic paradigm in investigating G is to find a nontrivial homomorphism $\varphi \colon G \to H$. One then either deduces a result (such as being of infinite order) from the quotient $\varphi(G)$ itself, or finds a suitable (finite index) subgroup $U \leq G$ as pre-image under φ of a subgroup of $\varphi(G)$. By transitioning to a presentation of U this process can be iterated. To pull results back to G, a homomorphism $\psi \colon U \to Q$ on U yields an induced representation $\psi\uparrow^G \colon G \to Q \wr S_{[G:U]}$ into a wreath product, that is defined on all of G.

2.2 Searching for Generator Images

The elementary way for finding homomorphisms is simply to take a finite group H, and to try out all possibilities for images for the generators of G in H on whether they yield a homomorphism, using van Dyck's theorem. This can be done in a backtrack-style search. It often is desirable to demand surjectivity (to reduce the number of images), and to consider such images only up to conjugacy in H. The resulting algorithm is called `Homomorphisms`, or `GQuotients`. It is particularly useful to identify (almost) simple groups H as quotients of G, since for these groups tend to have the smallest number of candidate images up to conjugacy.

Such a search takes naively $|H|^n$ steps (where n is the rank of G) and conjugation does not fundamentally reduce this cost. In practice this means that the scope of such a search is prospective quotients of order $\sim 10^7$ to 10^8.

A significantly more sophisticated approach [PF09] determines geometric conditions on the entries of potential matrix images of the generators of G, thus determining all q for which $PSL_2(q)$ could be a quotients of G. This algorithm is currently not implemented in Magma, but not in GAP.

2.3 Low Index Methods

An interesting variant is to take as image groups H the symmetric groups S_i up a given degree $d \geq i$, but instead of surjectivity require the image to be transitive. The point stabilizers in these images then correspond exactly to subgroups of G of index $i \leq d$, which explains why this variant is called the “Low Index Algorithm”.

The cost of such a calculation grows with $d!^n$. This means that the scope of such calculations is often $\lesssim 20$ and even in favorable cases rarely more than 35.

In certain combinatorial applications one needs to find not all subgroups, but only normal subgroups, though for significantly larger indices – possibly 10^4. Following an approach from [Fir05], this is done easiest through a combination of `GQuotients` for almost simple groups and suitable quotient algorithms applied to subgroups. Such an algorithm is implemented in GAP through the `LINS` package [Rob23].

2.4 Permutation action on subgroup cosets

If $U \leq G$ is a subgroup of index d, the action of G on the cosets of U will yield a homomorphism $G \to S_d$. In the case that U is given through a generating set, the coset enumeration process by Todd and Coxeter [TC36], the oldest algorithm for groups, can be used to determine the permutation image of such a homomorphism. (This, in particular, also determines the index $d = [G : U]$ and thus can be used, in the case of $U = \langle 1 \rangle$, to determine the order of G.)

The process works by systematically enumerating cosets of U, starting with the trivial coset $U \cdot 1$, as images of known cosets under generators and their inverses. Two cosets get identified, if group relation or subgroup generators force them to be equal. The construction of a new coset $U \cdot xg$, as image of the coset Ux under the generator g, not only establishes a coset representative xg, but also provides the permutation action of the generator g on the coset. Once the process finishes (i.e. all cosets have defined images under all generators, and no cosets remain forced to coincide), the calculation thus obtains permutation images for generators and their inverses in the permutation action φ of G on the cosets of U.

(The matrix, whose columns are the permutation images is called the *coset table.* For working with the subgroup, however the associated homomorphism is more useful.) The subgroup U then can be represented as the pre-image, under φ, of a point stabilizer in $\varphi(G)$.

The scope of such an enumeration is in the range of 10^6 to 10^7.

2.5 Quotient Algorithms

The last class of algorithms find quotients of G, that are the largest within a particular class. A prototype of this is the *abelianization* of G (the largest abelian quotient). It is obtained by writing the relators abelianized in rows of a matrix and transforming it into Smith Normal Form.

More generally, quotient algorithms take an existing (WLOG surjective) quotient $\varphi\colon G \to Q$ and determine a maximal Q-module M (for a chosen prime p, or even the ring of integers), subject to predefined constraints, such that there is a homomorphism $\psi\colon G \to M.Q$ through which φ factors.

By iterating this caonstructs quotients of G that are maximal as p-groups (or nilpotent groups) of user-defined class (The p-quotient algorithm, PQ, [NO96]), solvable of defined extension structure (solvable quotient, SQ, [Ple87, Nie94, Brü98]), or simply extend a given finite quotient φ (hybrid quotient, [DH21]). There also are variants for infinite polycyclic quotients.

The suitable module M and extension $M.Q$ are found by constructing, in a process similar to constructing all groups of a given order, a maximal *cover group* $K.Q$ as an extension of an module K by Q that still can be generated by the same number of elements. The algorithm then evaluates the relators in R in this extension, resulting in a normal subgroup $N \leq K$ and yielding $M = K/N$.

The scope of the p-Quotient algorithm can be quotients of order $p^1 00$; the other algorithms typically are more typically quotients of order 10^7 to 10^9.

3 Word Gymnastics

Products of group elements that occur in an abstract proof look very much like words in a free group. It thus is natural to expect calculations in finitely presented groups to consist of manipulating words, using the relations that define the group. Indeed such algorithms exist but are typically less impactful than one might imagine, since it is often impossible to create rules or recipes from which the right sequence of manipulations follows.

What exists are processes that fundamentally replace substrings of words in a systematic way. In computer science, this is called *string rewriting.*

3.1 Normal Forms

Many of the difficulties in handling finitely presented groups would disappear, if we could use the defining relations to bring every word representing a group element into a *normal form* that only depends on the element of G it represents. This is the case in textbook examples for certain prominent classes of groups (such as dihedral groups or Coxeter groups). Alas, effective generalization is hard. One way for this is to use the Knuth-Bendix [KB70] algorithm (considering the group as a monoid by taking inverses as new generators) in the hope of determining a finite confluent rewriting system for it. Indeed, the powerful techniques for polycyclic presentations or `PcGroups` [HEO05, Chapter 8] are a special case.

In `GAP`, the use of a (length-based) normal form for `FPGroups` can be triggered for a group G by the command `SetReducedMultiplication(G);` Arithmetic done after this command will bring elements into this normal form. It works very well for small groups, but can run into difficulties for orders beyond 10^5.

```
gap> f:=FreeGroup("r","s");; G:=f/ParseRelators(f,"r5,s2,sr=Rs");;
gap> r:=G.1;; s:=G.2;; (r^7)^(s^4);
s^-4*r^7*s^4;
gap> SetReducedMultiplication(G); # trigger reduction
gap> (r^7)^(s^4);
r^-1
```

In general, however, such a system is not guaranteed to exist if the group is infinite, furthermore there are no bounds on memory requirements, or on run time required. This difficulty, together with the need to define a suitable ordering, puts more sophisticated uses of this tool beyond the scope of this article.

We note, however, that such rewriting systems and resulting normal forms can provide a very elegant way of proving that a group is infinite, if the shape of the rules allow for the deduction of infinitely many possible normal forms, see for example [HH10].

3.2 Reidemeister Rewriting

The Reidemeister-Schreier rewriting process [MKS66, § 2.3] provides an effective way of realizing a subgroup of finite index as a finitely presented group on its own. The generators used are the Schreier generators of the subgroup.

It's input is a finitely presented group G, and a permutation representation of G (given through images of the generators) on the cosets of a subgroup $U \leq G$.

This representation defines, through lexicography on words, representatives for the cosets of the subgroup, and thus Schreier generators. By definition, the product or a coset representative by a generator $r_1 \cdot g = s \cdot r_2$ can be written as product of a Schreier generator s with another coset representative r_2. Tracing a word $w \in U$, given in the generators of G, through the cosets thus accumulates a factorization of w into product of Schreier generators.

Tracing through the conjugates $r \cdot w \cdot r^{-1}$, where w runs through relators and r through coset representatives, gives defining relators for a presentation of U in the Schreier generators. A few remarks on this process are in order:

1. Generator number and total relation size increase roughly by a factor of $[G : U]$. This puts the scope of the general method at around index 10^4 to 10^5.
2. There are some Schreier generators and some rewritten relators that can be guaranteed a priory to be trivial, or duplicate. The *Reduced Reidemeister-Schreier* process incorporates this in the process to obtaining a somewhat reduced presentation.
3. If we are only interested in in the structure of the abelianization U/U', we can save storage by treating the rewritten relators as commutative words.
4. Another variant, called *Modified Todd-Coxeter* (MTC), produces a presentation in a user-selected set of generators, at the cost of a less nice presentation.
5. An important special case of the MTC is that of a cyclic subgroup. Here any rewritten relator is just a power of the generator. This is used in the standard algorithm for determining the order of a finite finitely presented group: Find a cyclic subgroup of (moderate – say 10^5 to 10^6) finite index. Rewrite the presentation to this subgroup. The whole group order then is the product of subgroup index and subgroup order.

3.3 Tietze Transformations

It often turns out that there is a significant amount of redundancy in the presentations resulting from Reidemeister rewriting. It thus is desirable [HKRR84] to reduce these presentations in a greedy manner through a sequence of suitable Tietze transformations that can eliminate generators or reduce relators:

– Delete relators that are duplicate or conjugates
– If a relators has form $r_1 = a \cdot b \cdot c$ with the length of b being larger than that of $a^{-1} \cdot c^{-1}$, replace any occurrence of b in other relators by $a^{-1} \cdot c^{-1}$.
– If a generator occurs in only one relator, use it to replace this generator (and thereby the total number of generators used).
– Duplicate (or conjugate) relators obviously can be deleted.

Elimination of generators often leads to a significant growth of the overall presentation length, thus some heuristic on generator reduction needs to be used.

Often such a reduction is done automatically as part of a rewriting routine, in this case generator reduction is not as aggressive as it would be with an explicit call for reduction.

In GAP such rewriting processes with subsequent Tietze reduction are typically encapsulated within a homomorphism, for example `IsomorphismFpGroup` which provides a map to an isomorphic group with the new presentation, respectively `IsomorphismFpGroupByGenerators` which does so for a desired generating set.

When asking for properties of a subgroup (such as quotients), the system internally calculates such a homomorphism, and then works in its image.

3.4 Extending a Quotient

By iteratively considering quotients of subgroups, it is often possible to obtain larger quotients of a finitely presented group through an induced subgroup representation. This is particularly useful for the case of abelian quotients of subgroups (for which the task of rewriting the presentation is easier): Given a finitely presented group G, and a homomorphism $\varphi\colon G \to Q$, consider a subgroup $\ker\varphi \le U < G$ with $S = \varphi(U)$. (In fact U will have been obtained as pre-image of the subgroup $S < G$.) If $S/S' \not\cong U/U'$, we can find a subgroup $U' \le V < U$ such that $\ker\varphi \not\le V$. (The question for which subgroups one can guarantee this property is studied in [Hul01].)

The action of G on the cosets of then V is the induced representation of the action of U on the cosets of V. The defining quotient homomorphism of $V \cap \ker\varphi$ has a larger image. GAP implements a algorithm for finding such a subgroup V (with suitable defining quotient homomorphism) through the operation `LargerQuotientBySubgroupAbelianization`.

The scope of this method is similar to that of rewriting a presentation, about index $[G : U] \lesssim 10^4$.

4 Examples

We illustrate these algorithms in three examples, taken from joint work with Gerhard Rosenberger. We start with a group for which we prove, in three standard ways, that it is infinite.

Consider the tetrahedron group (from [FHH$^+$08] where it is handled through a purely theoretical argument):

$$\langle x, y, z \mid x^3 = y^3 = z^5 = (xyx^2y^2)^2 = (yz^3)^2 = (xz)^2 = 1\rangle.$$

```
gap> F:=FreeGroup("x","y","z");;
gap> rels:=ParseRelators(F,"x3=y3=z5=(xyx2y2)2=(yz3)2=(xz)2=1");
[ x^3, y^3, z^5, (x*z)^2, (y*z^3)^2, (x*y*x^2*y^2)^2 ]
gap> G:=F/rels;;
gap> AbelianInvariants(G);
[  ]
```

Note that G is perfect, and thus has no nontrivial solvable quotients.

Method 1: Subgroups of Small Index

In about 3 minutes we calculate representatives the 28 subgroup classes of index up to 15. None of it has infinite abelianization (commutator factor group U/U').

```
gap> l:=LowIndexSubgroupsFpGroup(G,15);;
gap> List(l,AbelianInvariants);
[ [  ], [ 3 ], [ 3 ], [ 3 ], [ 3 ], [ 2, 2, 4 ], [ 2 ], [  ], [...]
```

Index 16 takes 10 minutes, and finds 8 more subgroups, none of which prove infinity. Index 17 finds no further subgroups, but takes about half an hour. Larger indices thus get quickly out-of-reach.

Instead, we intersect the subgroups of index up to 15. It gives a quotient of structure $2^{10}.(A_5^4 \times A_6)$.

```
gap> k:=Intersection(l);;
gap> q:=DefiningQuotientHomomorphism(k);; p:=Image(q);
<permutation group of size 4777574400000 with 3 generators>
gap> Size(RadicalGroup(p)); # largest solvable normal
1024
gap> StructureDescription(p/RadicalGroup(p));
"A6 x A5 x A5 x A5 x A5"
```

To find subgroups of moderate index, we search through the 2nd maximal subgroups (maximals and maximals thereof) of this image by increasing index. At index 20 we happen to find a subgroup, whose pre-image in G has infinite abelianization, thus proving that G is infinite.

```
gap> m:=LowLayerSubgroups(p,2);;SortBy(m,x->Index(p,x));
gap> u:=First(m,x->0 in AbelianInvariants(PreImage(q,x)));;Index(p,u);
20
gap> AbelianInvariants(PreImage(q,u));
[ 0, 2, 2 ]
```

This subgroup in fact arises already in a much smaller quotient $2^5.A_6$.

```
gap> uq:=Image(FactorCosetAction(p,u));;StructureDescription(uq);
"(C2 x C2 x C2 x C2 x C2) : A6"
gap> MinimalFaithfulPermutationDegree(uq);
12
```

But finding this quotient directly would be hard, since G has a quotient $2^{61}.A_6$

```
gap> q:=GQuotients(G,AlternatingGroup(6));;
gap> Collected(AbelianInvariants(Kernel(q[1])));
[ [ 2, 61 ] ]
```

Method 2: Simple Quotients

There are other successful ways to prove infinity. For example we could iterate through simple groups (more generally: also subgroups of their automorphism groups), trying to find those that are quotients of G:

```
gap> it:=SimpleGroupsIterator();;
gap> repeat p:=NextIterator(it);q:=GQuotients(G,p);until Length(q)>0;p;
A5
```

Repeating this process finds, iteratively, quotients of type A_6, $PSp_4(4)$ and $PSp_6(2)$. If we search through the subgroups of this last group, we find a subgroup of index 56, whose pre-image in G has an infinite abelian quotient.

```
gap> q:=q[1];;p:=Image(q);
PSp(6,2)
gap> m:=List(ConjugacyClassesSubgroups(p),Representative);;Length(m);
1369
gap> SortBy(m,x->Index(p,x));
gap> u:=First(m,x->0 in AbelianInvariants(PreImage(q,x)));;Index(p,u);
56
gap> AbelianInvariants(PreImage(q,u));
```

```
[ 0, 0, 0 ]
```

Method 3: Existence of p-group Quotients

Alternatively once more, we can use the infinity criterion from [New90], which is based on a sharpened form [GN70] of the Golod-Shafarevich theorem.

For this we rewrite the presentation to the kernel of the projection from G onto A_6. We temporarily set the informational output higher to see the numbers on which the result is based.

```
gap> q:=GQuotients(G,AlternatingGroup(6));
[ [ x, y, z ] -> [ (1,4,2), (1,2,4)(3,6,5), (1,2,3,4,5) ] ]
gap> K:=Kernel(q[1]);;
gap> SetInfoLevel(InfoFpGroup,1);
gap> NewmanInfinityCriterion(K,2);
#I  61 generators, 360 relators, p=2, d=61 e=1531
#I  infinite by criterion 2
true
```

We find that this subgroup K has a (rewritten) presentation on $b = 61$ generators and $s = 360$ relators. Setting $K_1 = [K, K]K^2$ and $K_2 = [K_1, K]K^p$ and $d = \mathrm{rank} K/K_1$, $e = \mathrm{rank} K_1/K_2$. By [Joh97, 2^{nd} edition, Proposition 16.3, p.198] (the paper [New90] does not spell out the formula for $p = 2$), the group K thus has arbitrarily large 2-quotients and thus is infinite.

4.1 Finding Free Subgroups

Next, we consider the group [FHgR+11, (16) on p.12]

$$G = \langle x, y, z \mid x^3 = y^4 = z^3 = (xyxy^3)^2 = (yz)^2 = (xz)^2 = 1\rangle$$

Our goal is to expose a subgroup which has a free quotient of rank > 1 (which proves the existence of a free subgroup). This involves a further investigation of a suitable subgroup with an infinite abelian quotient of rank > 1.

Again we start by searching for subgroups of small index, looking for some that have an infinite abelianization.

```
gap> f:=FreeGroup("x","y,"z");;
gap> G:=f/ParseRelators(f,"x3=y4=z3=(xyxy3)2=(yz)2=(xz)2=1");;
gap> l:=LowIndexSubgroupsFpGroup(G,25);;
gap> l:=Filtered(l,x->0 in AbelianInvariants(x));;
```

It turns out that all abelianizations of these subgroups have a torsion component. We pick a subgroup for which the torsion seems small, and calculate again subgroups of it.

```
gap> u:=First(l,x->AbelianInvariants(x)=[0,0,2]);
Group(<fp, no generators known>)
gap> Index(G,u);
24
gap> l2:=LowIndexSubgroups(u,2);
gap> List(l2,AbelianInvariants);
[ [ 0, 0, 2 ], [ 0, 0, 2, 2 ], [ 0, 0, 0, 0 ], [ 0, 0, 2, 2 ],
  [ 0, 0, 4, 8 ], [ 0, 0, 2, 4 ], [ 0, 0, 0, 2, 2 ], [ 0, 0, 2, 4 ] ]
gap> v:=l2[3];;Index(g,v); # number 3 has abelianization [0,0,0,0]
48
```

We find a subgroup (of index 48 in the original group) whose abelianization is a free abelian group. This is our candidate. We rewrite the presentation and find a group on 4 generators, which however is not free.

```
gap> iso:=IsomorphismFpGroup(v);;h:=Range(iso);
<fp group on the generators [ F1, F2, F3, F4 ]>
gap> RelatorsOfFpGroup(h);
[ F2*F3*F2^-2*F3^-1*F2, F3*F1^2*F3^-1*F1^-2,[...]
```

In the hope of identifying a free quotient of lower rank, we recall that a free group of rank 2 has [Hal36] exactly 19 quotients of type A_5 and 53 of type A_6:

```
gap> Length(GQuotients(FreeGroup(2),AlternatingGroup(5)));
19
gap> Length(GQuotients(FreeGroup(2),AlternatingGroup(6)));
53
```

Inspecting quotients $H \twoheadrightarrow A_5$, we find exactly 19 that map the first and second generator in the same way, and all of these map the 3rd generator trivially:

```
gap> q:=GQuotients(h,AlternatingGroup(5));;Length(q);
28
gap> tst:=Filtered(q,x->Image(x,h.1)=Image(x,h.2));;Length(tst);
19
gap> List(tst,x->ImagesRepresentative(x,h.3));
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]
```

Similarly, there are 53 quotients to A_6 that satisfy these two conditions:

```
gap> q:=GQuotients(h,AlternatingGroup(6));;
gap> Number(q,x->Image(x,h.1)=Image(x,h.2) and IsOne(Image(x,h.3)));
53
```

This indicates that the quotient obtained by factoring out the quotient of the first two generators, as well as the third one, might be free of rank 2. We are able to verify this by constructing this quotient and simplifying its presentation [2]

```
gap> f:=h/[h.1/h.2,h.3];;sim:=IsomorphismSimplifiedFpGroup(f);
[ F1, F2, F3, F4 ] -> [ F1, F1, <identity ...>, F4 ]
gap> Range(sim);
<fp group of size infinity on the generators [ F1, F4 ]>
gap> RelatorsOfFpGroup(Range(sim));
[  ]
```

4.2 Proving Finiteness

For a final example, we show that a group is finite. The example taken is (this is [FHH+08, p. 560, Nr. 17])

$$G = \langle x, y \mid x^2, y^3, z^2, (xyxyxyxy^2xy^2xyxy^2xy^2)^2, (yz)^2, (xz)^2 \rangle.$$

Again, we start looking at subgroups of small index. Trying increasing index limits we find that we are able to calculate these to a comparatively large index, but that none has an infinite abelian quotient.

```
gap> f:=FreeGroup("x","y","z");; G:=f/ParseRelators(f,
> "x2,y3,z2,(xyxyxyxy2xy2xyxy2xy2)2,(yz)2,(xz)2");;
gap> l:=LowIndexSubgroups(G,60);;Length(l);
621
gap> ForAny(l,x->0 in AbelianInvariants(x));
false
gap> Collected(List(l,x->Length(GeneratorsOfGroup(x))));
[ [ 2, 27 ], [ 3, 151 ], [ 4, 253 ], [ 5, 103 ], [ 6, 63 ], [ 7, 21 ],
  [ 8, 2 ], [ 9, 1 ] ]
```

2 Caveat: in GAP the command `IsFreeGroup` only indicates whether a group was created as a free group and is not a proper freeness test.

Furthermore, we see that most of these subgroups can be generated (using the Reduced Reidemeister-Schreier method) by few elements. All of this indicates that the group might be finite.

However an attempt to calculate the order of G directly ultimately fails for reasons of memory usage in a Todd-Coxeter coset enumeration:

```
gap> Size(G);
#I  Coset table calculation failed -- trying with bigger table limit
#I  Coset table calculation failed -- trying with bigger table limit
[... message repeats a number of times, until ultimately: ]
Error, reached the pre-set memory limit  in
TCENUM.CosetTableFromGensAndRels( fgens, grels, fsgens) at [...]
```

What happens here is that the standard test for finiteness in GAP (see item 5 in section 3.2) tries to find a cyclic subgroup of moderate index. We shall see that in this example the largest cyclic subgroup has index $\sim 7 \cdot 10^6$, which makes this standard approach unsuitable.

Instead, we seek a quotient of the group that is as large as we can find (in the example it is in fact a faithful representation) and in this quotient look for a (possibly non-cyclic) subgroup of index in the range $10^4 - 10^5$. We rewrite the presentation to the pre-image of this subgroup (and hope it does not get too large) and then try to show (by enumerating cosets of a cyclic subgroup) that the group given by this rewritten presentation is finite.

We thus investigate the quotient exposed by the subgroups we found so far:

```
gap> k:=Intersection(l);
Group(<fp, no generators known>)
gap> q:=DefiningQuotientHomomorphism(k);;
gap> p:=Image(q);
<permutation group of size 13271040 with 3 generators>
```

We calculate second maximal subgroups of this quotient (we tried maximal ones first, but they did not help), and try to use their abelianizations to find a larger quotient (as described in section 3.4):

```
gap> m:=LowLayerSubgroups(p,2);;Length(m);
87
gap> subs:=List(m,x->LargerQuotientBySubgroupAbelianization(q,x));;
gap> subs:=Filtered(subs,x->x<>fail);; Length(subs);
12
```

```
gap> k:=Intersection(k,Intersection(subs));;
gap> q:=DefiningQuotientHomomorphism(k);;p:=Image(q);
<permutation group of size 849346560 with 3 generators>
```

Repeating this (and trying other quotients) does not yield a larger quotient.

We notice that a subgroup of this quotient of order a few 1000s will have index a few 10000s, and thus would be amenable for finding a presentation by rewriting.

We try a chain of subgroups containing the normalizer of the 3-Sylow and attempt maximals and second maximal subgroups in a subgroup of index about $4 \cdot 10^4$. (There are many other ways of finding a suitable subgroup.)

```
gap> u:=Normalizer(p,SylowSubgroup(p,3));
<permutation group of size 5184 with 5 generators>
gap> ac:=AscendingChain(p,u);;List(ac,Size);
[ 5184, 20736, 82944, 5308416, 21233664, 212336640, 849346560 ]
gap> m:=LowLayerSubgroups(ac[2],2);;
gap> Collected(List(m,Size));
[ [ 1728, 3 ], [ 2304, 5 ], [ 2592, 7 ], [ 3456, 32 ], [ 5184, 36 ],
  [ 6912, 3 ], [ 10368, 7 ], [ 20736, 1 ] ]
```

We settle on a subgroup of order 6912, and rewrite the presentation to its pre-image in G:

```
gap> u:=First(m,x->Size(x)=6912);
<permutation group of size 6912 with 4 generators>
gap> v:=PreImage(q,u);
Group(<fp, no generators known>)
gap> iso:=IsomorphismFpGroup(v);;
gap> h:=Range(iso);
<fp group on the generators [ F1, F2, F3, F4, F5 ]>
```

We try calculating the order of this finitely presented group, and find it to be 6912. This proves that G is finite of order $|v| \cdot [G : v] = 849346560$. Incidentally this also proves that the quotient q we found is a faithful permutation representation.

```
gap> Size(h);
6912
gap> Size(h)*Index(g,v);
849346560
```

This representation allows us to identify the structure of the group as being a central product of $SL_2(5)$ with a large solvable group.

```
gap> n:=Filtered(NormalSubgroups(p),x->Size(x)<Size(p) and Size(x)>1);;
gap> m:=Filtered(Combinations(n,2),x->p=ClosureGroup(x[1],x[2])
> and Size(Intersection(x[1],x[2]))<4);
[ [ <permutation group of size 14155776 with 13 generators>,
      <permutation group of size 120 with 2 generators> ]]
gap> Size(Intersection(m[1]));
2
gap> StructureDescription(m[1][2]);
"SL(2,5)"
gap> IsSolvableGroup(m[1][1]);
true
```

(We also can calculate that the largest cyclic subgroup of G has order 120, indicating that indeed the approach of calculating the order through a large cyclic subgroup would be infeasible in this example.)

4.3 Epilogue

While these are selected examples, they are not atypical. In the author's experience, examining broader sets of groups with presentations of similar characteristics, the success rate of determining finiteness with the methods presented here has been clearly upwards of 70%. Not bad for a task that cannot have a general solution. Of course, things become more difficult or even impossible, if questions hinge on establishing indices (or group orders) that are clearly beyond 10^9.

In investigating finitely presented groups, the computer therefore is never a replacement for a human researcher, but just a tool that helps in settling many cases, allowing for the human to concentrate on what cases are difficult and thus worthy of investigation with human ingenuity.

Literatur

[Atk84] Michael D. Atkinson, editor. *Computational group theory.* Academic press, 1984.

[BCP97] W. Bosma, J. Cannon, and C. Playoust. The MAGMA algebra system I: The user language. *J. Symbolic Comput.*, 24(3/4):235–265, 1997.

[Boo57] William Boone. Certain simple, unsolvable problems of group theory. VI. *Nederl. Akad. Wet., Proc., Ser. A*, 60:227–232, 1957.

[Brü98] Herbert Brückner. *Algorithmen für endliche auflösbare Gruppen und Anwendungen.* Dissertation, Rheinisch-Westfalische Technische Hochschule, Aachen, Germany, 1998.

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

[FHgR$^+$11] Benjamin Fine, Alexander Hulpke, Volkmar große Rebel, Gerhard Rosenberger, and Stefanie Schauerte. The Tits alternative for short generalized tetrahedron groups. *Scientia Series A: Math.Sciences*, 21:1–15, 2011.

[FHH$^+$08] Benjamin Fine, Miriam Hahn, Alexander Hulpke, Volkmar große Rebel, Gerhard Rosenberger, and Martin Scheer. All finite generalized tetrahedron groups. *Algebra Colloq.*, 15(4):555–580, 2008.

[Fir05] David Firth. *An Algorithm to Find Normal Subgroups of a Finitely Presented Group, up to a Given Finite Index.* PhD thesis, Warwick University, 2005.

[GN70] W. Gaschütz and M. F. Newman. On presentations of finite p-groups. *Journal für die Reine und Angewandte Mathematik*, 245:172–176, 1970.

[GAP] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.12.* http://www.gap-system.org, 2022.

[Hal36] P. Hall. The Eulerian functions of a group. *Q. J. Math., Oxf. Ser.*, 7:134–151, 1936.

[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.

[HH10] George Havas and Derek F. Holt. On Coxeter's families of group presentations. *J. Algebra*, 324(5):1076–1082, 2010.

[HKRR84] George Havas, P. E. Kenne, J. S. Richardson, and E. F. Robertson. A Tietze transformation program. In Atkinson [Atk84], pages 69–73.

[Hul01] Alexander Hulpke. Representing subgroups of finitely presented groups by quotient subgroups. *Experiment. Math.*, 10(3):369–381, 2001.

[Joh97] D. L. Johnson. *Presentations of groups.*, volume 15 of *Lond. Math.*

Soc. Stud. Texts. Cambridge: Cambridge University Press, 2nd ed. edition, 1997.

[KB70] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In John Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon press, 1970.

[LNS84] Reinhard Laue, Joachim Neubüser, and Ulrich Schoenwaelder. Algorithms for finite soluble groups and the SOGOS system. In Atkinson [Atk84], pages 105–135.

[MKS66] Wilhelm Magnus, Abraham Karrass, and Donald Solitar. *Combinatorial group theory: Presentations of groups in terms of generators and relations.* Interscience Publishers [John Wiley & Sons, Inc.], 1966.

[Neu82] J. Neubüser. An elementary introduction to coset table methods in computational group theory. In Colin M. Campbell and Edmund F. Robertson, editors, *Groups – St Andrews 1981, Proceedings Conference St Andrews, 1981*, volume 71 of *London Mathematical Society Lecture Note Series*, pages 1–45. Cambridge University Press, 1982.

[New90] M. F. Newman. Proving a group infinite. *Arch. Math. (Basel)*, 54(3):209–211, 1990.

[Nie94] Alice C. Niemeyer. A finite soluble quotient algorithm. *J. Symbolic Comput.*, 18(6):541–561, 1994.

[NO96] M. F. Newman and E. A. O'Brien. Application of computers to questions like those of Burnside. II. *Internat. J. Algebra Comput.*, 6(5):593–605, 1996.

[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.

[PF09] W. Plesken and A. Fabiańska. An L_2-quotient algorithm for finitely presented groups. *J. Algebra*, 322(3):914–935, 2009.

[Ple87] W. Plesken. Towards a soluble quotient algorithm. *J. Symbolic Comput.*, 4(1):111–122, 1987.

[Rob23] F. Rober. LINS, provides an algorithm for computing the normal subgroups of a finitely presented group up to some given index bound., Version 0.6. , 2023. GAP package.

[Sim94] Charles C. Sims. *Computation with finitely presented groups.* Cambridge University Press, 1994.

[TC36] J.A. Todd and H.S.M. Coxeter. A practical method for enumerating cosets of a finite abstract group. *Proc. Edinburgh Math. Soc.*, 5:26–34, 1936.