Last Time

Orbit / Stabilizer algorithm

Decompose group: Subgroup and Cosets

Use for permutation groups with subgroup being point stabilizer.

Can iterate again for subgroups: Logarithmize problem.

Stabilizer Chains

Let $G \leq S_\Omega$. A list of points $B=(\beta_1,\ldots,\beta_m)$, $\beta_i \in \Omega$ is called a ***base***, if the identity is the only element $g \in G$ such that $\beta_i^g=\beta_i$ for all i.

The associated ***Stabilizer Chain*** is the sequence

$$G=G^{(0)} > G^{(1)} > \ldots > G^{(m)}= \langle 1 \rangle$$

defined by $G^{(0)}:=G$, $G^{(i)}:=\mathrm{Stab}_{G^{(i-1)}}(\beta_i)$. (Base property guarantees that $G^{(m)}= \langle 1 \rangle$.)

Note that every $g \in G$ is defined uniquely by base images $\beta_1^g,\ldots,\beta_m^g$. (If g,h have same images, then g/h fixes base.)

Data structure

We will store for a stabilizer chain:

- The base points $(\beta_1,\ldots,\beta_m)$.
- Generators for all stabilizers $G^{(i)}$. (Union of all generators is ***strong generating set***, as it permits reconstruction of the $G^{(i)}$.) Data structure thus is often called **B**ase and **S**trong **G**enerating **S**et.
- The orbit of β_i under $G^{(i-1)}$ and an associated transversal for $G^{(i)}$ in $G^{(i-1)}$ (possibly as *Schreier tree*).

Storage cost thus is $\mathcal{O}(m \cdot |\Omega|)$

Data structure

We will store for a stabilizer chain:

- The base points $(\beta_1,\ldots,\beta_m)$.
- Generat… f… of all gener… permits recons… thus is often called … Strating Set.
- The orbit of β_i under $G^{(i-1)}$ and an associated transversal for $G^{(i)}$ in $G^{(i-1)}$ (possibly as *Schreier tree*).

Storage cost thus is $\mathcal{O}(m \cdot |\Omega|)$

Construction: Later

Consequences

- Group order: $G = [G^{(0)}:G^{(1)}]\cdot\ldots[G^{(m-1)}:G^{(m)}]$ and thus $G=\prod_i |\beta_i{}^{G(i-1)}|$.
- Membership test in G for $x \in S_\Omega$:

1. Is $\omega = \beta_1{}^x \in \beta_1{}^G$? If not, terminate.
2. If so, find transversal element $t \in G^{(0)}$ such that $\beta_1{}^t=\beta_1{}^x$.
3. Recursively: Is x/t (stabilizing β_1) in $G^{(1)}$.
   (Respectively: test x/y=( ) in last step.)

More Consequences

Bijection $g \in G \Leftrightarrow$ base image $(\beta_1{}^g, \beta_2{}^g, \ldots)$.

- Enumerate G, equal distribution random elements.
- Write $g \in G$ as product in transversal elts.
- Write $g \in G$ as product in strong generators.
- If strong gens. are words in group gens.: Write $g \in G$ in generators of G. (Caveat: Often long words)
- Chosen base: Find stabilizers, transporter elements, for point tuples.

More Consequences

Bijection $g \in G \Leftrightarrow$ base image $(\beta_1{}^g, \beta_2{}^g, \ldots)$.

- Enumerate G, equal distribution random elements.
- Write g
- Write $g \in G$ as product in strong generators.

Computable Bijection $G \leftrightarrow \{1,2,\ldots,|G|\}$

- If strong gens. are words in group gens.: Write $g \in G$ in generators of G. (Caveat: Often long words)
- Chosen base: Find stabilizers, transporter elements, for point tuples.

Schreier-Sims algorithm

Sims' (~1970) idea is to use a membership test in a partially completed stabilizer chain to reduce on the number

Basic stru ze
$\leq G^{(i-1)}$ g d
β^U with t ro
generato

The basic pa
$G^{(i-1)}$ to t on
image $\omega = \beta^x$.

Charles Sims, *1938

Otto Schreier, 1901-1929

Schreier-Sims algorithm

Sims' (~1970) idea is to use a membership test in a partially completed stabilizer chain to reduce on the number of Schreier generators.

Basic structure is a partial stabilizer, i.e. a subgroup $U \leq G^{(i-1)}$ given by generators and a base-point orbit β^U with transversal elements (products of the generators of U).

The basic operation now is to pass an element $x \in G^{(i-1)}$ to this structure and to consider the base point image $\omega = \beta^x$.

Base point image $\omega = \beta^x$

- If $\omega \in \beta^U$, transversal element $t \in U$ such that $\beta^t = \omega$. Pass $y = x/t$ to the next lower partial stabilizer $\leq G^{(i)}$.

  If $\omega \notin \beta^U$, add x to the generating set for U and extend the orbit of β. All new Schreier generators $y \in \mathrm{Stab}_U(\beta)$ are passed to next partial stab. $\leq G^{(i)}$.

  If no lower stabilizer was known, test whether the y was the identity. If so just return. (Successful membership test.)

  Otherwise start new stabilizer for generator y and the next base point. (Pick a point moved by y).

Homomorphisms

Embed permutation group G into direct product $D=G\times H$.
A homomorphism $\varphi:G\to H$ can be represented as $U\leq D$ via

$$U=\{(g,h) \in G\times H \mid g\varphi=h\}$$

Build a stabilizer chain for U using only the G-part.

Then decomposing $g \in G$ using this chain produces an H-part that is $g^{(\varphi-1)}$.
Use this to evaluate arbitrary homomorphisms.

Kernels, Relators

Vice versa, let $\varphi: H \to G$ and
$U = \{(g,h) \in G \times H \mid h\varphi = g\}$.

Form a stabilizer chain from generators of U, using the G-part.
The elements sifting through this chain (trivial g-part) are generators for $\ker \varphi$.

If H is a free group, this yields relators (later) of a presentation for G.

Other Actions

Every finite group is a permutation group in suitable actions. (E.g. matrices on vectors.) Same methods apply there.

It is possible to use different actions (e.g. matrix group on subspaces and on vectors)

But: Orbit lengths can be unavoidably huge if there are no subgroups of small index.

Approach can be useful for well-behaved groups. Not a panacea, but part of matrix group recognition.

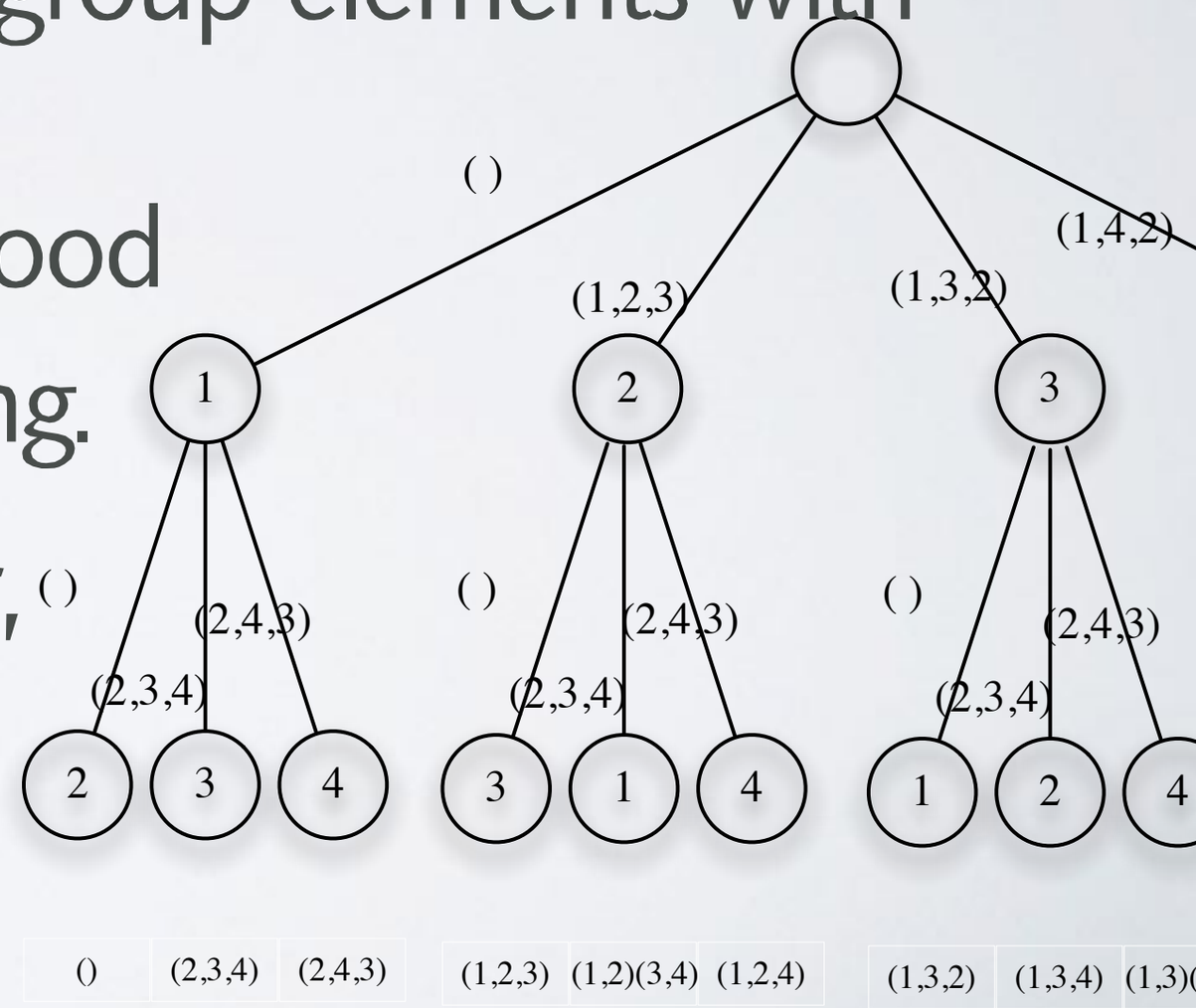
Permutation Backtrack

A stabilizer chain lets us consider the elements of G as leafs on a tree, branches corresponding to base point images.

Traverse the tree (depth first) by enumerating all possible base images. Find group elements with particular desired property.

Exponential run time but good in practice, by clever pruning.

E.g.: Centralizer, Normalizer, Set Stab., Intersection, conj.elts., ...



In GAP

Permutation groups use backtrack search for

- Centralizer, element conjugacy `RepresentativeAction`
- Normalizer, Subgroup conjugacy `RepresentativeAction`
- Set Stabilizer, Intersection,
- Generic: `ElementProperty`, `SubgroupProperty`

Quandry

Schreier-Sims is deterministic Algorithm. Polynomial (in the degree $n=|\Omega|$) runtime, but larger exponent (naively $\mathcal{O}(n^7)$, $\mathcal{O}(n^3)$ if Schreier tree is used cleverly).

The cause is the processing of all (mostly redundant) Schreier generators.

In practice this is not feasible if n is not small (>1000). For short base ($\log|G| \leq \log^c n$) we would like nearly linear time $\mathcal{O}(n \log^c n)$, best possible

Wrong Results are Cheap

Use only *some* generators (random subset, better: random subproducts). Potentially wrong data structure. But:

- Error results in chain that claims to be too small - can detect if group order is known. (Random-Schreier-Sims)
  - Assign a known group order (`SetSize(G,order);`)
- Error analysis: A random element of G fails sifting in wrong chain with probability 1/2 - guarantee arbitrary small error probability.
- Verify correctness, by showing the group cannot be larger:
  - Combinatorial Verification (SIMS, see SERESS' book)
  - Presentation from stabilizer chain. Verify that group fulfills it. If too small, some relators fail to be.

Wrong Results are Cheap

Use only *some* generators (random subset, better: random subproducts). Potentially wrong data structure. But:

- Error results in chain that claims to be too small - can detect if group order is known. (Random-Schreier-Sims)
- ➡ Assign a known group order (`SetSize(G,order);`)
- Error analysis: A random element of *G* fails sifting in wrong chain with probability 1/2 - guarantee arbitrary small error probability.
- Verify correctness, by showing the group cannot be larger.
- ➡ Combinatorial Verification (SIMS, see SERESS' book)
- ➡ Presentation from stabilizer chain. Verify that group fulfills it. If too small, some relators fail to be.

Present

Future

4

Finitely Presented Groups

Finitely Presented Group

Let F be a free group on finite alphabet A.

Let R (*Relators / Relations*) be a finite subset of F.

(Interpret $a=b$ as a/b.)

Then $\langle A \mid R \rangle = F/N$, where N is the smallest normal subgroup of F containing R, is a *finitely presented group*.

Some people distinguish the groups (quotients of F) and *presentations* (formal string objects) to describe modifications.

Example

First textbook example of a nonabelian group:

$$D_8 = \langle r, s \mid r^4 = s^2 = 1, rs = sr^{-1} \rangle$$

- Elements have normal form $s^a r^b$, $0 \leqq a \leqq 1$, $0 \leqq b \leqq 3$.
  - So group has at most 8 elements.
  - And indeed there are 8 different ones (show that elements have different actions).

Group order bounded by normal form. Can we find one?

Actions (homomorphisms) show we achieve that image.

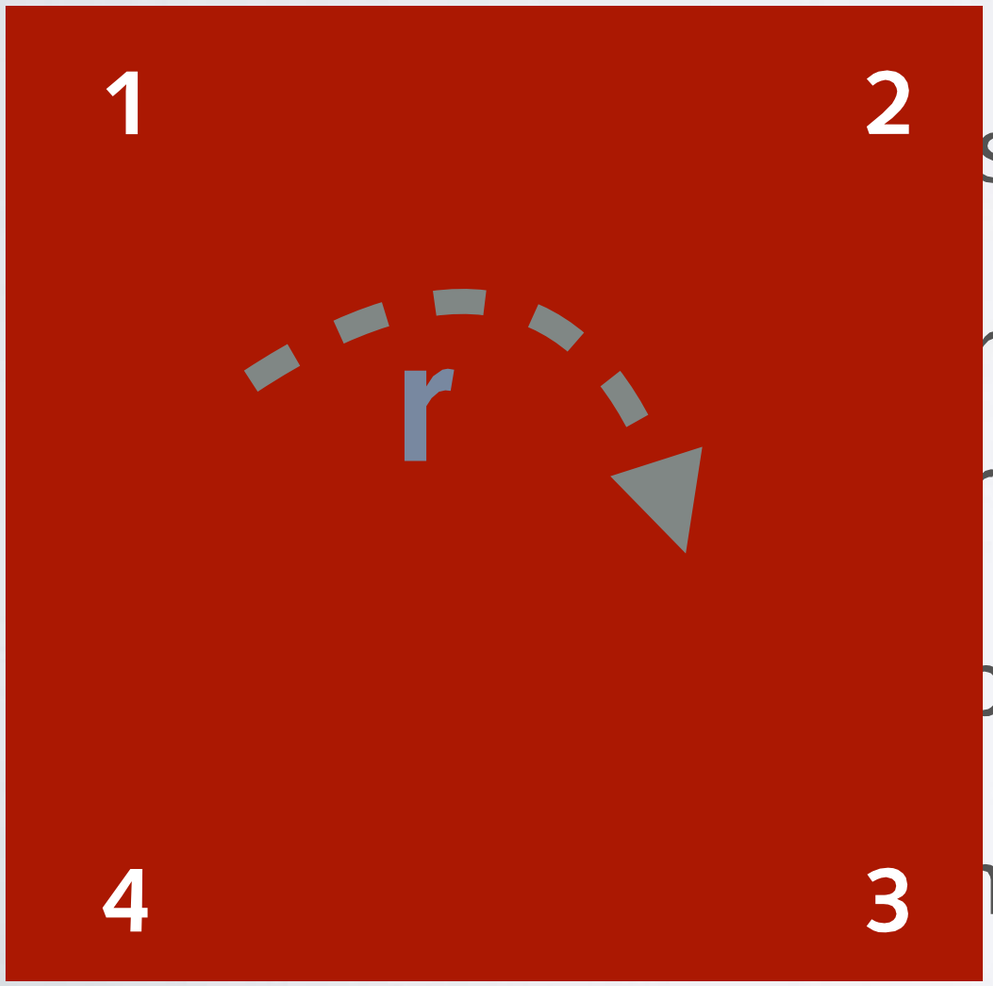
Example

First textbook example of [illegible] abelian group:

rotation

$$D_8 = \langle r, s \mid r^4 = s^2 = 1, rs = sr^{-1} \rangle$$

- Elements have normal form $s^a r^b$, $0 \leqq a \leqq 1$, $0 \leqq b \leqq 3$.



st Spiegelung (*ger.*) =reflection

re 8 different ones (show that

rent actions).

by normal form. Can we find one?

ms) show we achieve that image.

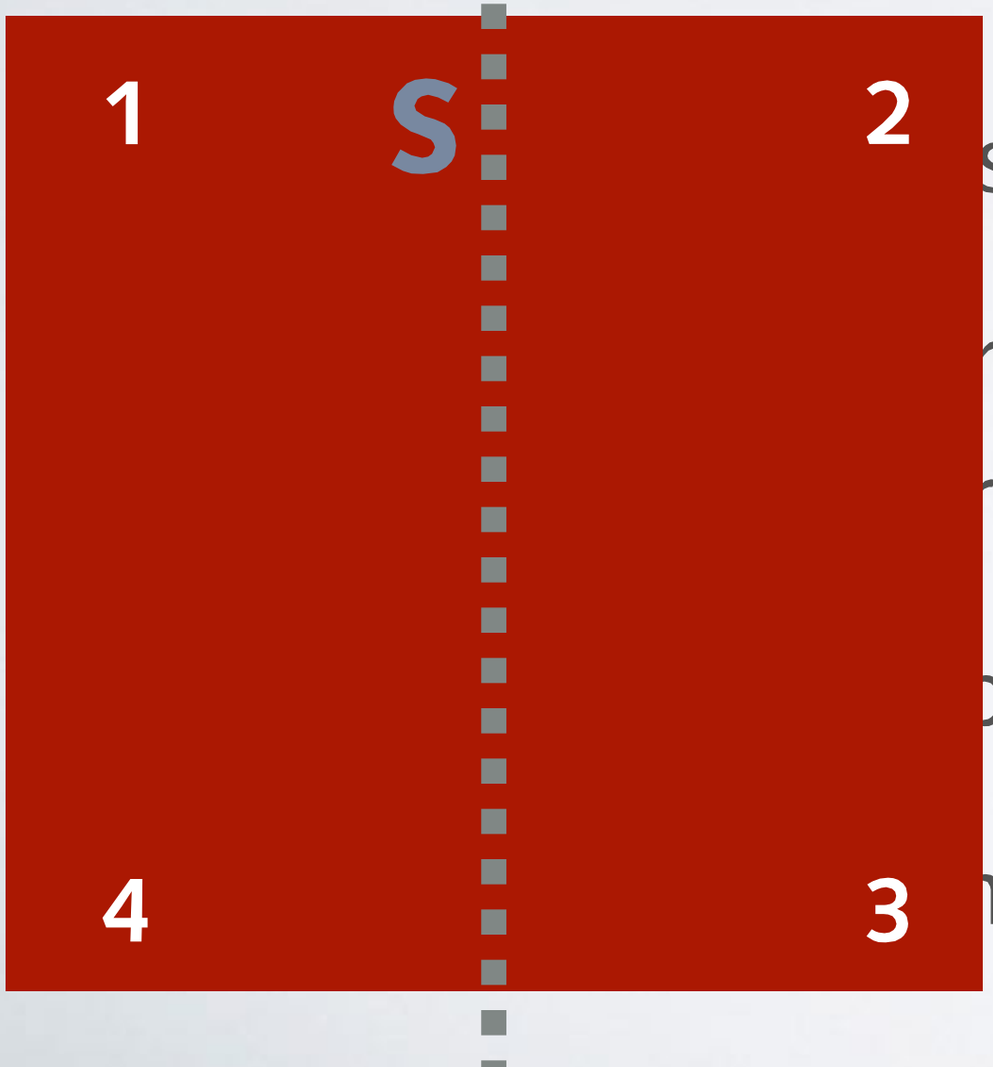
Example

First textbook example of [illegible] abelian group:

rotation

$$D_8 = \langle r, s \mid r^4 = s^2 = 1, rs = sr^{-1} \rangle$$

- Elements have normal form $s^a r^b$, $0 \leqq a \leqq 1$, $0 \leqq b \leqq 3$.



Spiegelung (*ger.*)
=reflection

st

re 8 different ones (show that

rent actions).

by normal form. Can we find one?

ms) show we achieve that image.

Example

First textbook example of a nonabelian group:

$$D_8 = \langle r, s \mid r^4 = s^2 = 1, rs = sr^{-1} \rangle$$

- Elements have normal form $s^a r^b$, $0 \leqq a \leqq 1$, $0 \leqq b \leqq 3$.
  - So group has at most 8 elements.
  - And indeed there are 8 different ones (show that elements have different actions).

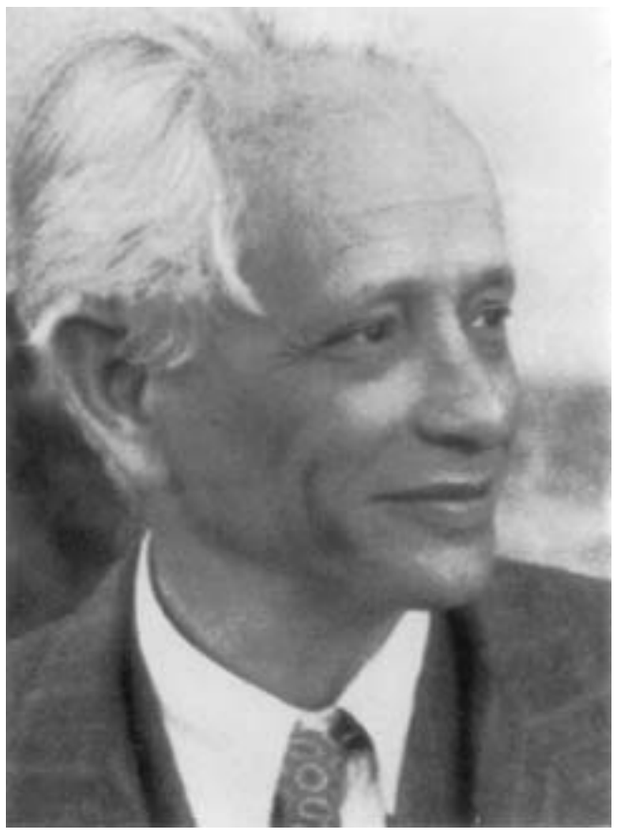
Group order bounded by normal form. Can we find one?

Actions (homomorphisms) show we achieve that image.

Dehn's Problems (1911)

Given a fi… …p, can we test algorith…

1. A w… …ity
2. Tw… …ements
3. Tes…



Max Dehn, 1878-1952

Dehn's Problems (1911)

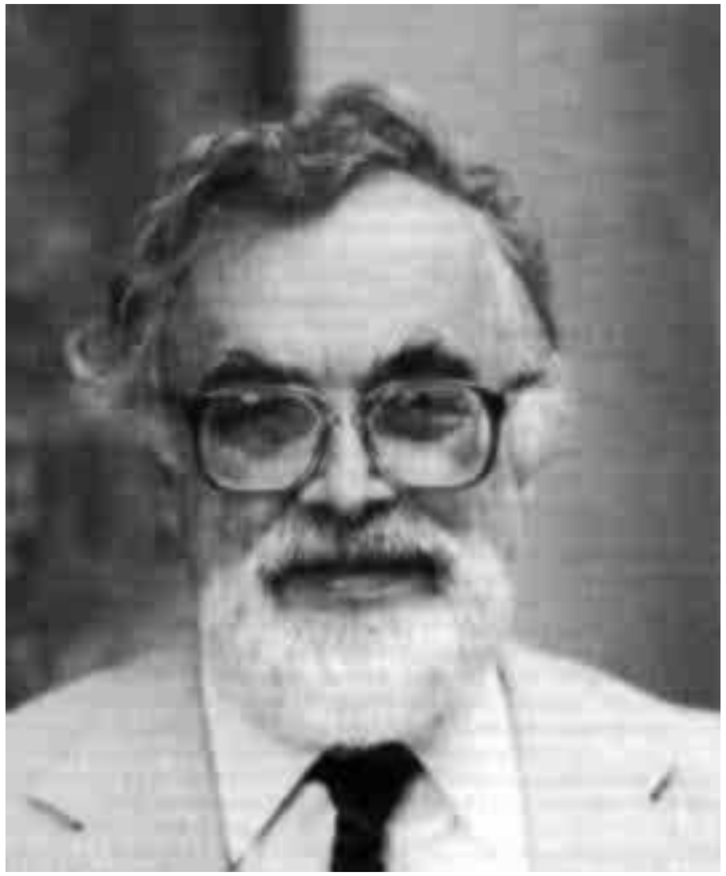
Given a finitely presented group, can we test algorithmically whether:

1. A word represents the identity
2. Two words are conjugate elements
3. Test for Group Isomorphism

ble…

Give… oup,
algo…

1. … entity
2. … e eler
3. Test fo… oup isomorphis…



William Boone, 1920-1983

Piotr Novikov, 1901-1975

Answer: Boone, Novikov (1957) No.

Reduce to Halteproblem for Turing machines — there is no universal machine. Any approach will be heuristics.

In GAP

Create a free group and a list of relators (elements of group), quotient is finitely presented group.

GeneratorsOfGroup or AssignGeneratorVariables allows access to generators — no automatic assignment.

Convenience function ParseRelators.

```
gap> f:=FreeGroup("a","b");
<free group on the generators [ a, b ]>
gap> rels:=ParseRelators(f,"a3,b5,(ab)2");
[ a^3, b^5, (a*b)^2 ]
gap> g:=f/rels;
<fp group on the generators [ a, b ]>
gap> AssignGeneratorVariables(g);
#I  Assigned the global variables [ a, b ]
gap> a in f; a in g;
false   /   true
```

Element Equality

Elements seemingly multiply as in the free group, but equality tests will test properly. The first such test for a group thus will trigger a highly nontrivial calculation.

One can also force reduction for all products formed.

```
gap> a*b*a*b;
(a*b)^2
gap> Order(a*b*a*b);
1
gap> a*b*a*b=One(g);
true
gap> a^7=a;
true
gap> a^8=a;
false
```

```
gap> g:=f/rels;;
gap> g.1=a;
false
gap> SetReducedMultiplication(g);
gap> AssignGeneratorVariables(g);
#I  Variable `a' will be overwritten
#I  Variable `b' will be overwritten
gap> a^-1*b^5*a^7*b^3;
b^-2
```

Element Equality

Elements seemingly multiply as in the free group, but equality tests will test properly. The first such test for a group thus will trigger a highly nontrivial calculation.

One can also force reduction for all products formed.

Either a faithful permutation representation, or a *confluent rewriting system* — generalization of the idea of collection for arbitrary groups.

```
gap> g:=f/rels;;
gap> g.1=a;
false
gap> SetReducedMultiplication(g);
gap> AssignGeneratorVariables(g);
#I  Variable `a' will be overwritten
#I  Variable `b' will be overwritten
gap> a^-1*b^5*a^7*b^3;
b^-2
```

Homomorphisms

If $\langle A \mid R \rangle$ is an f.p. group, a map given on the generators extends to a homomorphism, iff the generator images satisfy the relators.

(Free group: Any such map extends to homomorphism — analog to basis property in linear algebra.)

The main tool for working with f.p. groups thus is to work with homomorphisms.

E.g. p-Quotient algorithm (Heiko Dietrich's talks), `AbelianInvariants` / `MaximalAbelianQuotient`

G-Quotient / Low Index

Given a f.p. group *G* and a finite group *H*, search for all homomorphisms from *G* to *H:*

Test all tuples of elements of *H* for whether they satisfy the relators — then they can be images for *G*'s generators.

Test only up to conjugacy, might want onto map.

Variant: *H* symmetric group of degree *n*, no onto requirement: Find (pre-images of point stabilizer) all subgroups of *G* of index up to *n*: ***Low Index*** algorithm.

```
gap> f:=FreeGroup("r","s");;
gap> g:=f/ParseRelators(f,"r3,s4,(rs)6,[r,s]2");;
gap> Size(g);
23040

gap> g:=f/ParseRelators(f,"r3,s4,(rs)10,[r,s]2");
gap> Size(g);   CTRL-C after a while
Error, user interrupt in
brk> CTRL-D
gap> AbelianInvariants(g);
[ 2 ]
gap> l:=LowIndexSubgroupsFpGroup(g,30);
[ Group(<fp, no generators known>), …
gap> List(l,x->Index(g,x));
[1,2,6,12,15,30,30,30,30,30,30,30,10,…
gap> List(l,AbelianInvariants);
[[2],[],[2],[],[2,2,2],[2,2,2],[2,2],[2,2],
[2,2,2],[0,2],[2,2],[2,2,3],[2,4],[2,4],[2,4],
[2,2,2],[2,2,2],[2,2,4],[2,2,2,2],[0,4],[2,2],
[2],[2],[2,2,2,2],[2,2,4],[2],[2,2,2,2,3],
[2,2,8],[]]
```

```
gap> f:=FreeGroup(“a","b","c");;
gap> g:=f/ParseRelators(f,
> “a3,b7,c19,(ab)2,(bc)2,(ca)2,(abc)2");;
gap> it:=SimpleGroupsIterator();;
gap> repeat h:=NextIterator(it); Print(“Gp:”,h,”\n”);
> q:=GQuotients(g,h); until Length(q)>0;Length(q);
Gp:A5
Gp:PSL(2,7)[…]
Gp:J_1
1
gap> k:=Kernel(q[1]);
Group(<fp, no generators known>)
gap> Index(g,k);
175560
gap> repeat h:=NextIterator(it); […]
Gp:PSL(2,113)
gap> k2:=Kernel(q[1]);;
```

```
gap> f:=FreeGroup("a","b","c");;
gap> g:=f/ParseRelators(f,
> "a3,b7,c19,(ab)2,(bc)2,(ca)2,(abc)2");;
gap> it:=SimpleGroupsIterator();;
gap> repeat h:=NextIterator(it); Print("Gp:",h,"\n");
> q:=GQuotients(g,h); until Length(q)>0;Length(q);
Gp:A5
Gp:PSL(2,7)[…]
Gp:J_1
1
gap> k:=Kernel(q[1]);
Group(<fp, no generat
gap> Index(g,k);
175560
gap> repeat h:=NextIterator(it); […]
Gp:PSL(2,113)
gap> k2:=Kernel(q[1]);;
```

Parameters: Starting Order. Potentially also Option :nopsl2, e.g. SimpleGroupsIterator(60:nopsl2).

Representing Subgroups

Work with homomorphisms to represent subgroup $U<G$ as pair $(\varphi, U\varphi)$ with $\varphi: G \to P$ for a (finite, permutation) group P. Can do well

- Normalizer, (Normal) closure
- Intersection (Embed in direct product)
- Subgroup of Subgroup (Embed in wreath product)
- pre-images of arbitrary subgroups

```
gap> s:=Intersection(k,k2);
Group(<fp, no generators known>)
gap> Index(g,s);
126647579520
gap> q:=DefiningQuotientHomomorphism(s);
[ a, b, c ] ->
[ (1,260,241)(2,240,177)(3,91,130)(4,234,203)
gap> p:=Image(q);
<permutation group of size 126647579520 with 3 generators>
gap> m:=MaximalSubgroupClassReps(p);;List(m,x->Index(p,x));
[ 114, 6328, 6441, 30058, 30058, 266, 1045, […]
gap> u:=PreImage(q,m[4]);
Group(<fp, no generators known>)
gap> Index(g,u);
30058
gap> AbelianInvariants(u);
[ 2 ]
gap> AbelianInvariants(m[4]);
[ 2 ]
```