Rewriting systems for a family of perfect groups

Jack Schmidt

University of Kentucky

AMS CGT 2007-03-03

Rewriting systems are used to efficiently find and analyze families of perfect groups analogous to p-groups of maximal class. Computer calculations are quite effective with this method and are used to give evidence and counterexamples to various generalizations of the theory of p-groups of maximal class to these families of perfect groups.

Overview

- I began studying some new groups that seemed similar to some old groups.
- There are good ideas and software for the old groups, but the old software does not work for the new groups.
- I made new software to apply the old ideas to the new groups.
- The new software shows the new groups really are different, and probably need new ideas too.

Key points about the new groups

- The new groups have a very natural definition
- The definition is almost identical to coclass for p-groups (the old groups)
- The new groups have some nice properties very similar to the properties of the old groups, specifically the coclass tree and uniserial action
- Therefore, one should try to mimic the old calculations and find some partial coclass trees

Nilpotent normal subgroups to build the group

- Groups G can be understood as built from G/N and N, and we even assume N is nilpotent.

- Groups without nilpotent normal subgroups are tabulated up to very large orders. For small orders ($< 10^{10}$) the perfect ones are direct products of simple groups.

- The unique largest nilpotent normal subgroup is called the Fitting subgroup, and is denoted $Fit(G)$.

- Nilpotent groups are too hard to handle all at once

Modules to build the normal nilpotent subgroup

- All nilpotent minimal normal subgroups are $\mathbb{Z}[G/Fit(G)]$-modules.
- There is a unique largest subgroup of G that is a $\mathbb{Z}[G/Fit(G)]$-module, namely the center of $Fit(G)$.
- G is a repeated downward extension of $G/Fit(G)$ by $\mathbb{Z}[G/Fit(G)]$-modules, namely the factors of the upper central series of $Fit(G)$.
- $\mathbb{Z}[G/Fit(G)]$ is fixed, but has complicated modules
- My family is defined by requiring we use only simple modules

What if the modules are simple?

- If they are simple modules, then $Fit(G)$ is a p-group
- The upper central and lower central series are equal
- In fact all characteristic subgroups are in that series
- So a sort of uniserial action of $G/Fit(G)$ on $Fit(G)$
- We say G/N is the parent of G, and this forms a tree with the original $G/Fit(G)$ as a root (be careful of $Fit(G/N) \neq Fit(G)/N$)
- Surely these trees are infinite with short, periodic limbs all coming off one infinite branch which defines a nice uniserial action on a p-adic group of some sort?

Summary of the groups

- Natural definition linking the module and commutator structure of the Fitting subgroup
- Nice properties similar to coclass for p-groups
- Studied together as a tree, and the p-group case is very well studied
- Should expect entire family to be described by a single infinite group constructed from (very many) repeated extensions by simple modules

Key points for rewriting systems

- Old software fails due to inappropriate data-type (one cannot handle perfect groups, one cannot handle extensions)
- Rewriting systems generalize pc-presentation to more groups
- Handle extensions very well, especially by nilpotent subgroups
- Allows my “new” algorithm for efficient calculation of isomorphism classes, modeled after pc-presentation algorithm
- Much faster for the generalized coclass trees than the old software for permutation groups

What are rewriting systems?

- Formalize what it means to **simplify** in a finitely presented group
- Elements of a group are represented as formal products of generators X, so an epimorphism $\phi : X^* \to G$ takes formal **words** and multiplies them.
- If $\phi(x) = \phi(y)$ represent the same element, which should we use, x or y, to represent the element?
- There is no general answer for finitely presented groups
- Rewriting systems are a systematic answer to this question, and always exist for finite groups

Definition of simplest words and rules

- Define an ordering on the free monoid, such that $x < y$ if $\phi(x) = \phi(y)$ and we prefer x to y
- We should also prefer *axb* to *ayb*.
- Should be well-ordered, so there is a **simplest** word for every $\phi(x)$
- Replacing *ayb* by *axb* is symbolized by the rule $y \mapsto x$.
- The official **rules** of the rewriting system are $\{y \mapsto x : \phi(y) = \phi(x), x < y$, x and every proper subword of y are simplest words $\}$
- Necessary and sufficient to reduce any word to its simplest form

Simplest words for extensions

- If $\phi : X^* \to G/N$ and $\phi : Y^* \to N$ have been used to form rewriting systems for G/N and N, then we can define a $\phi : (X \cup Y)^* \to G$ as well
- Since $Ng = gN$, we can rewrite yx to xy' and group all the ys together.
- Define an ordering on $(X \cup Y)^*$ so that $yx > xy'$. The standard way to do this is called the wreath product ordering.
- The simplest words of G are then just xy where x is a simplest word for G/N and y is a simplest word for N.

Rules and tails for extensions

- The rules for G are
  1. The unchanged rules for N
  2. $yx \mapsto xy'$ describing the action of G/N on N
  3. Modified rules for G/N: $x \mapsto x'$ becomes $x \mapsto x'y$ where y is the simplest word for $\phi(x')^{-1}\phi(x) \in N$
- The ys in the third type of rules are called **tails**.
- An extension is defined by:
  1. the rules of G/N,
  2. the rules of N,
  3. the action (rules) of G/N on N, and
  4. the tails, a function from $Rules(G/N)$ to N.

Not all tails work

- Not every element of $N^{Rules(G/N)}$ defines a downward extension of G/N by N.
- Difference between **final** forms of words and simplest form
- Minimal word with non-simplest final form is ABC with $AB \mapsto R$ and $BC \mapsto S$ rules, but RC and AS don't have a common final form
- If these **overlaps** are fine, then all words are fine.

Overlapping rules

- Most overlaps in an extension work out automatically:

  1. N with N work out, because N is a group
  2. N with action rules work out, because each element of G/N acts as an automorphism of N,
  3. Action rules with G/N work out, because the map from G/N to Aut(N) is a homomorphism.
  4. N and G/N don't overlap

- Only overlaps left are G/N with G/N and these must always agree on the G/N part; only the N part of the final form can differ.
- All final forms are xy, with x and y simplest forms for G/N and N
- In the extension, all final forms must be equal, so it defines a quotient N/K instead of N.

Checking overlaps is linear algebra

- Need to simplify products in X^* as if they were in G.
- Applying $x_i \mapsto x_i' \; y_i$:

$$ax_i b \rightarrow ax_i' y_i b \rightarrow ax_i' b \; y_i^b$$

Where y_i^b is the simplest form for $\phi(y_i)^{\phi(b)} \in N$.

- Many applications like this still give words of the form $xy_1^{m1} y_2^{m2} \cdots$ where y_i are tails, and m_i are in group ring of G/N
- Setting two final forms equal only requires the tails y_i to be in the kernels of the differences $m_i - m_i'$.
- Finding tails is just a giant null space calculation

Isomorphism testing is also easy

- Checking overlaps finds $Z^2(G/N, N)$, want $H^2 = Z^2/B^2$ instead
- Finding $B^2(G, V)$ is very easy; "Fox derivative"
- Still isomorphic groups that are not isomorphic as extensions; orbits of stabilizer of V in $Aut(G/N)$ on $H^2(G/N, N)$
- Algorithm constructs $Aut(G)$ while computing orbits and from first cohomology

Comparison with other algorithms

Method:	Factor Sets	Polycyclic	Subgroups	Rewriting
Person:	Schreier	Eick	Holt	Schmidt
Groups:	Finite	Polycyclic	Finite	Finite
Input:	AsSet	Pc-Pres	Perm	Rws
Output:	AsSet	Pc-Pres	Fp-group	Rws
Time:	poly(G)	polylog(G)	polylog(G/H)	polylog(G)

- Notice difference in input/output data types
- Subgroup chains can be added to the rewriting algorithm, but only reduces constants, not complexity

Conclusion

- Rewriting systems are abstractly nice for iterated extensions
- Polynomial algorithm for generation and isomorphism testing (requires arithmetic oracles, and has high startup cost).
- Will be available as a GAP package later this year
- Already used to compute coclass trees to depth 3 for all simple groups of order less than 1000, and very deep tree for $A_7 \mod 2$
- Some trees are finite! Some appear to have multiple trunks!

THE END