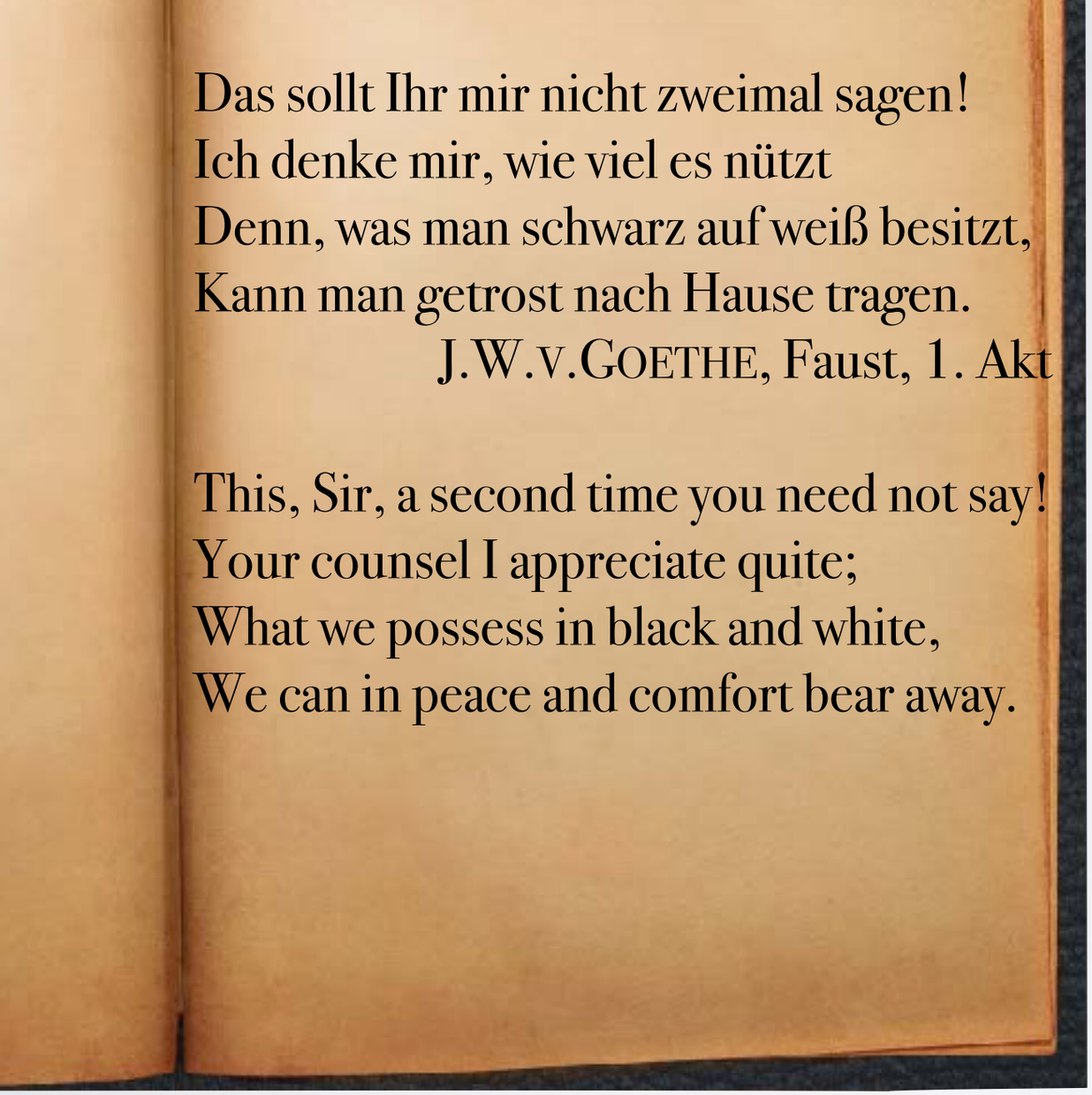
What Can Composition Trees Do For ~~Me~~ You

Alexander Hulpke
Department of Mathematics
Colorado State University
Fort Collins, CO, 80523, USA
http://hulpke.com

Partially supported by
NSF-DMS #1720146

Aachen, July 2019


Das sollt Ihr mir nicht zweimal sagen!
Ich denke mir, wie viel es nützt
Denn, was man schwarz auf weiß besitzt,
Kann man getrost nach Hause tragen.
J.W.v.GOETHE, Faust, 1. Akt

This, Sir, a second time you need not say!
Your counsel I appreciate quite;
What we possess in black and white,
We can in peace and comfort bear away.

Slides at

http://www.math.colostate.edu/~hulpke/talks/CT19.pdf

Executive Summary

I want composition tree to help work with matrix groups.

But, *even more*, I want the *parts* of composition tree to help with the performance of many calculations (which might seem to have nothing to do with matrix groups).

No, we're not selling composition tree for scrap

State law loophole allows stolen cars to be sold for scrap

By **AJ Giardina** | February 14, 2012 at 10:04 PM CST - Updated June 26 at 2:47 PM

GULFPORT, MS (WLOX) - Did you know when someone takes a car to a scrap metal yard or junk yard to sell, that person doesn't need t car title if the vehicle is at least ten years o

Some victims of car theft called WLOX complaining that because of this law, t never see their cars again. They want the current state law to prevent othe becoming victims.

exact," said Ashley C

Composition Tree Is

- A data structure for matrix groups
- Divide and conquer paradigm for arbitrary groups
- The algorithms used to build the data structure
- An implementation (existing, future, hypothetical)

of such algorithms in GAP

Composition Tree Is

- A data structure for matrix groups
- Divide and conquer paradigm for arbitrary groups
- The algorithms used to build the data structure
- An implementation (existing, future, hypothetical)

of such algorithms in GAP

Not a solitaire, but the combination of algorithms that are useful in their own.
Intermediate steps (even if only for small cases) are meaningful.
Pay-off not only when all implementation is done.

View Point

Imagine we already had composition tree (in all its glory) in GAP.

- **What would we use it for ?**
- **Which design questions do we need to answer ?**
- **What issues will turn up ?**

Hope this might help to guide implementation.

View Point

Imagine we already had composition tree (in all its glory) in GAP.

- **What would we use it for ?**
- **Which design q**
- **What issues wil**

Hope this might h

Using the parts of composition tree more will provide more tests, even without constructing complicated recognition test cases.

Immediate Consequences

A composition tree lets us compute

- Group Order
- Composition Structure
- Membership test
- Decomposition into generators — Evaluate Homomorphisms

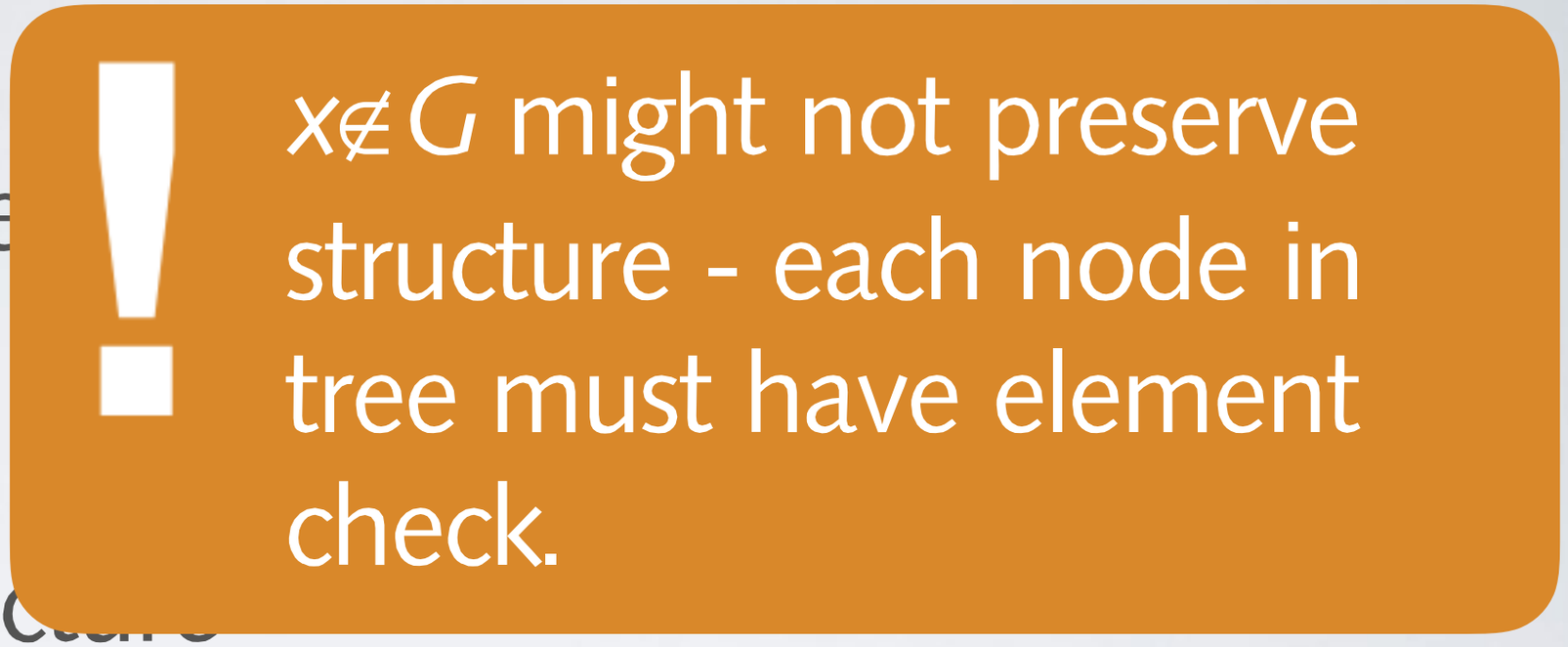
for matrix groups over finite fields.

Immediate Consequences

A composition tree

- Group Order
- Composition Structure
- Membership test
- Decomposition into generators — Evaluate Homomorphisms

for matrix groups over finite fields.



Immediate Consequences

A composition tree lets us compute

- Group Order
- Composition Structure
- Membership test
- Decomposition into generators — Evaluate Homomorphisms

for matrix groups over finite fields.

Immediate Consequences

A composition tree

- Group Order
- Composition Structure
- Membership test
- Decomposition into generators — Evaluate Homomorphisms

for matrix groups over finite fields.

Do we need "*strong*" generators into which one decomposes easily ?

The CGT Stack

Your Own Calculations; `FindCounterexample`

Isomorphism tests, Data Libraries

Find Classes, Subgroups, Characters; Test Properties

Homomorphisms (constructive membership)

Group Order, Subgroup Membership

Element Arithmetic and Equality

Potential Issues

- General membership test at start of every user function becomes expensive.
- How to decide between using nice monomorphisms (automatic translation to permutation action on vectors) and composition tree?
- Data structure for subgroups? (➠Solvable Radical)

Pote

Should matrices carry membership in a parent object which is preserved by arithmetic?

- General members function becomes expensive.
- How to decide between using nice monomorphisms (automatic translation to permutation action on vectors) and composition tree?
- Data structure for subgroups? (➠Solvable Radical)

Pote

Do not set `HandledBy...` flag by default, but test. (At function call? At creation?)

- General membership test at start of every user function becomes expensive.
- How to decide between using nice monomorphisms (automatic translation to permutation action on vectors) and composition tree?
- Data structure for subgroups? (➠Solvable Radical)

Pot

! We cannot rely on composition tree, unless the tree is verified correct. (➠Presentations)

- General member
  function becomes expensive.
- How to decide between using nice
  monomorphisms (automatic translation to permutation action on vectors) and composition tree?
- Data structure for subgroups? (➠Solvable Radical)

Potential Issues

- General membership test at start of every user function becomes expensive.
- How to decide between using nice monomorphisms (automatic translation to permutation action on vectors) and composition tree?
- Data structure for subgroups? (➠Solvable Radical)

- Re owers are p^2), alr
- In ent if fini . (joi
- Fu ayers for

Other Rings

- Residue class modulo m: Layers for prime powers are additive: $(I+pA)(I+pB) \equiv I+p(A+B) \pmod{p^2}$, already in `matgrp` package.
- Integers: Consider congruence images. Sufficient if finite group or congruence subgroup property. (joint w/ DETINKO,FLANNERY).
- Function Fields: Approximations give similar layers for powers of t. Not implemented yet.

O

- Residue class mod… are additive: (I+*pA*)(I+*pB*) = I+*p*(*A*+*B*) (mod *p*²) already in matgrp package.

> **!** Matrix arithmetic over other rings is slow!

- Integers: Consider congruence images. Sufficient if finite group or congruence subgroup property. (joint w/ DETINKO,FLANNERY).
- Function Fields: Approximations give similar layers for powers of *t*. Not implemented yet.

Solvable Radical

Standard approach for permutation groups.

Let $R \triangleleft G$ be solvable radical (largest solvable normal subgroup). Then all nonabelian composition factors occur in G/R.

Theorem: Action of G on all nonabelian chief factors has kernel R, Image with good permutation representation.

Known Algorithm Paradigms

Direct Construction, Lookup
(write down the result)
<=*linear time*

Linear Algebra
good polynomial time

(Combinatorial) Search
exponential time
possible

Known Algorithm Paradigms

Goal: Minimize Combinatorial search, reduce to linear algebra + looking up information in library.

Linear Algebra
good polynomial time

Direct Construction, Lookup
(write down the result)
<=*linear time*

Solvable Radical

Let $M/N \cong T^m$ be a nonsolvable chief factor. Then the m copies of T show up in the composition tree.

We can calculate the action of G on M/N from the composition tree, without holding M/N. Image in $\mathrm{Aut}(T) \wr S_m$. Get effective homomorphism ϱ: $G \to G/R$.

Generators for R from presentation of image of ϱ.

PCGS for R from stabilizer chain. (Shortish orbits.)

Basic version implemented in `matgrp` package.

Operation `FittingFreeLiftSetup`.

Solvable Radical

Let $M/N \cong T^m$ be a nonsolvable chief factor. Then the m copies of T show up in the composition tree.

We can calculate the action of G on M/N from the composition tree, without holding M/N. Image in $\mathrm{Aut}(T) \wr S_m$. Get effective homomorphism ϱ: $G \to G/R$.

Gene… R f…

PCG… …m …

Basic… …mpl…

Ope… …ting…

Use presentations of simple groups, constructive recognition to get good perm rep. for radical factor.

Solvable Radical

Let $M/N \cong T^m$ be a nonsolvable chief factor. Then the m copies of T show up in the composition tree.

We can calculate the action of G on M/N from the composition tree, without holding M/N. Image in $\mathrm{Aut}(T) \wr S_m$. Get effective homomorphism ϱ: $G \to G/R$.

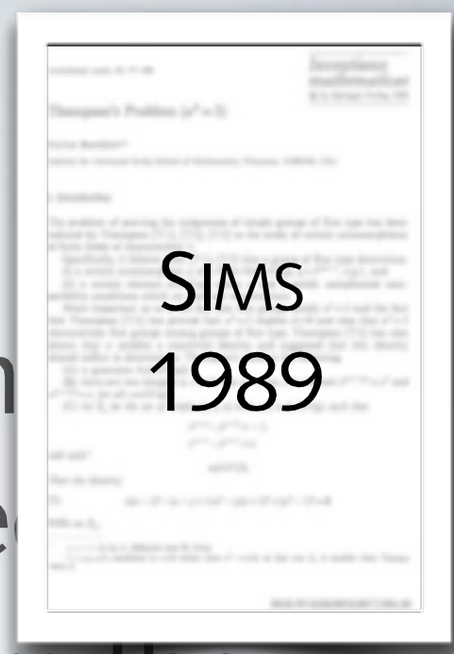
Generators for R from presentation of image of ϱ.

PCGS for R from stabilizer chain. (Shortish orbits.)

Basic version implemented in `matgrp` package.

Operation `FittingFreeLiftSetup`.

Solvable Radical



Let $M/N \cong T^m$ be a nonsolvable chief factor. Th

m copies of T show up in the composition tre

We can calculate the action of G on M/N from the composition tree, without holding M/N. Image in $\mathrm{Aut}(T) \wr S_m$. Get effective homomorphism ϱ: $G \to G/R$.

Generators for R from presentation of image of ϱ.

PCGS for R from stabilizer chain. (Shortish orbits.)

Basic version implemented in `matgrp` package.

Operation `FittingFreeLiftSetup`.

Let $M/N \cong$ Th
m copies re
We can c om the
compositi ge in
$\mathrm{Aut}(T) \wr S_m$. Get effective homomorphism ϱ: $G \to G/R$.

Get good base from composition tree (for R). Or use composition tree in place of StabChain? Analog of `SpecialPcgs`?

Generators for R from presentation of image of ϱ.
PCGS for R from stabilizer chain. (Shortish orbits.)
Basic version implemented in `matgrp` package.
Operation `FittingFreeLiftSetup`.

What Will Work Already?

Using existing (permutation group) methods in *G/R*:

- Conjugacy Classes, Centralizer, Element Conjugacy
- MaximalSubgroupClassReps

Not yet linked to standard function:

- Normalizer (`NormalizerViaRadical`)
- Hall (and thus Sylow) subgroups

In Principle, but not yet coded properly in library:

- Subgroup Lattice
- Automorphism Group

What Will Work Already?

Using existing (permutation group) methods in *G/R*:

- Conjugacy Classes, Centralizer, Element Conjugacy
- MaximalSubgroupClassReps

Not yet linked to standard function:

- Normalizer (`NormalizerViaRadical`)
- Hall (and thus Sylow) subgroups

In Principle, but

- Subgroup Latti
- Automorphism

Subgroups stored according to Radical data structure, same composition tree.

What W

But very basic for almost simple groups.
(➠Recognition)

Using existing (perm

- Conjugacy Classes, Centralizer, Element Conjugacy
- MaximalSubgroupClassReps

Not yet linked to standard function:

- Normalizer (`NormalizerViaRadical`)
- Hall (and thus Sylow) subgroups

In Principle, but not yet coded properly in library:

- Subgroup Lattice
- Automorphism Group

What W

Using existing (perr

- Conjugacy Classe
- MaximalSubgroupClassReps

Not yet linked to standard function:

- Normalizer (`NormalizerViaRadical`)
- Hall (and thus Sylow) subgroups

In Principle, but not yet coded properly in library:

- Subgroup Lattice
- Automorphism Group

Can be better than backtrack for some permutation groups, but how to select strategy?

What W

Using existing (perm

- Conjugacy Classe
- MaximalSubgroupClassReps

Not yet linked to standard function:

- Normalizer (`NormalizerViaRadical`)
- Hall (and thus Sylow) subgroups

In Principle, but not yet coded properly in library:

- Subgroup Lattice
- Automorphism Group

! Obstacle is basically to have a way to represent homomorphisms using composition tree

What W[illegible] Already?

Using exist[illegible]ds in *G/R*:

- Conjugac[illegible] Conjugacy
- Maxim[illegible]

Not ye[illegible]

- Norma[illegible]
- Hall (a[illegible]

In Princip[illegible] library:

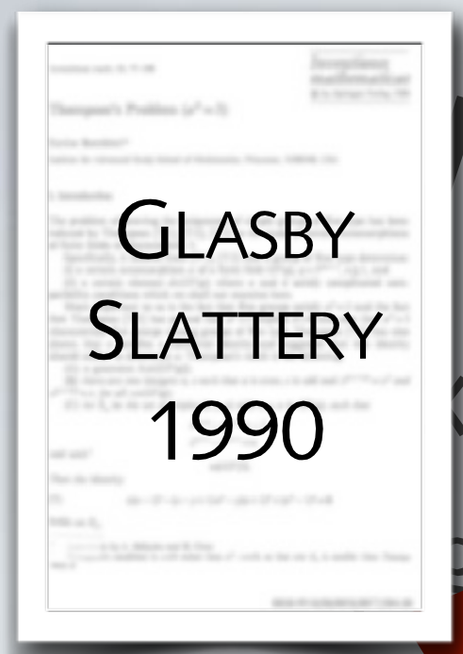
- Subgroup La[illegible]
- Automorphism G[illegible]p

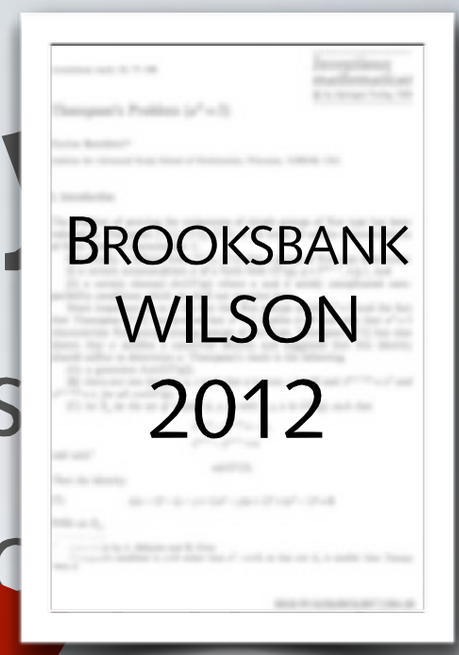
Not Yet attempted:

Possible Project: Intersection

Use:

- PcGroup ideas
- Existing Normalizer code
- Backtrack
- Intersection of Classicals





What W… Already…

- Maxim…
- Norma…
- Hall (a…

Not ye…

In Principl… library:

- Subgroup La…
- Automorphism G…p

Not Yet attempted:

Possible Project: Intersection

Use:

- PcGroup ideas
- Existing Normalizer code
- Backtrack
- Intersection of Classicals

```
gap> LoadPackage("matgrp"); #use recog
    [...]
gap> g:=AtlasSubgroup("Co1",IsMatrixGroup,3); #2^11.M24
<matrix group of size 501397585920 with 2 generators>
gap> FittingFreeLiftSetup(g); #10 seconds
rec( depths:=[ 1, 12 ],
  factorhom:=[ [<24x24 matrix GF2>, [...] ]] ->[ (1,11[...]],
  pcgs := Pcgs([ <24x24 matrix GF2>, [...] ]),
  pcisom := Pcgs([...]) -> Pcgs([ f1, f2, [...]]),
  radical := <matrix group size 2048 with 11 generators>,)
gap> Length(ConjugacyClasses(g)); #<1 second
80
gap> me:=ApplicableMethod(HallSubgroupOp,[g,[3]],0,3);;
gap> s:=me(g,[3]); #<0.1 second
<matrix group of size 27 with 3 generators>
gap> n:=NormalizerViaRadical(g,s); # 0.5 second
<matrix group of size 432 with 7 generators>
gap> m:=MaximalSubgroupClassReps(g);; # 2 seconds
gap> List(m,x->IndexNC(g,x));
[ 2048,24,276,759,1288,1771,2024,3795,40320,1457280 ]
```

```
gap> LoadP                    ); #use recog
    […]
gap> g:                       IsMatrixGroup,3); #2^11.M24
<matrix                       585920 with 2 generators>
gap> Fitt                     g); #10 seconds
rec( depths:=[ 1,
  factorhom:=[ [<24x24          atrix GF2>, […] ]] ->[ (1,11[…]],
  pcgs := Pcgs([ <24x24         trix GF2>, […] ]),
  pcisom := Pcgs([…]) -> P      gs([ f1, f2, […]]),
  radical := <matrix group      ize 2048 with 11 generators>,
gap> Length(ConjugacyClasses(g)); #<1 second
80
gap> me:=ApplicableMethod(HallSubgroupOp,[g,[3]],0,3);;
gap> s:=me(g,[3]); #<0.1 second
<matrix group of size 27 with 3 generators>
gap> n:=NormalizerViaRadical(g,s); # 0.5 second
<matrix group of size 432 with 7 generators>
gap> m:=MaximalSubgroupClassReps(g);; # 2 seconds
gap> List(m,x->IndexNC(g,x));
[ 2048,24,276,759,1288,1771,2024,3795,40320,1457280 ]
```

Cheat 1:
Avoid Nice
Monomorphism

```
gap> LoadP                    ); #use recog
    […]
gap> g:                       IsMatrixGroup,3); #2^11.M24
<matrix                       585920 with 2 generators>
gap> Fitti                    g); #10 seconds
rec( depths:=[ 1,
  factorhom:=[ [<24x24          atrix GF2>, […] ]] ->[ (1,11[…]],
  pcgs := Pcgs([ <24x24         trix GF2>, […] ]),
  pcisom := Pcgs[…]) -> Pcgs([ f1, f2, […]]),
  rad                 trix group size 2048 with 11 generators>,
ga                    gacyClasses(g)); #<1 second
8
ga                    leMethod(HallSubgroupOp,[g,[3]],0,3);;
gap                   ); #<0.1 second
<matri                of size 27 with 3 generators>
gap> n:=NormalizerViaRadical(g,s); # 0.5 second
<matrix group of size 432 with 7 generators>
gap> m:=MaximalSubgroupClassReps(g);; # 2 seconds
gap> List(m,x->IndexNC(g,x));
[ 2048,24,276,759,1288,1771,2024,3795,40320,1457280 ]
```

Cheat 1:
Avoid Nice
Monomorphism

Cheat 2:
Small Dimension

```
gap> LoadP                    ); #use recog
    […]
gap> g:                       IsMatrixGroup,3); #2^11.M24
<matrix                       585920 with 2 generators>
gap> Fitti                    g); #10 seconds
rec( depths:=[ 1,
  factorhom:=[ [<24x24          atrix GF2>, […] ]] ->[ (1,11[…]],
  pcgs := Pcgs([ <24x24         trix GF2>, […] ]),
  pcisom := Pcgs[…]) -> Pcgs([ f1, f2, […]])
  rad                 trix group size 2048 wit                >,
ga                    gacyClasses(g)); #<1
8
ga                    leMethod(HallSubgrou
gap                   ); #<0.1 second
<matri                of size 27 with 3 generato
gap> n:=NormalizerViaRadical(g,s); # 0.5 second
<matrix group of size 432 with 7 generators>
gap> m:=MaximalSubgroupClassReps(g);; # 2 seconds
gap> List(m,x->IndexNC(g,x));
[ 2048,24,276,759,1288,1771,2024,3795,40320,1457280 ]
```

Cheat 1:
Avoid Nice Monomorphism

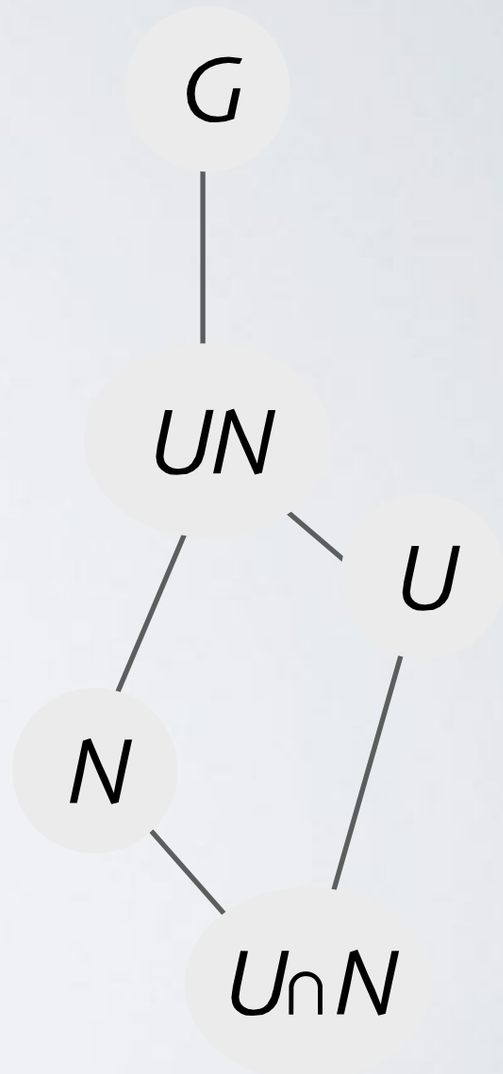
Cheat 2:
Small Dimension

Cheat 3:
Simple group of small degree, brute force isomorphism

Further Out

- Smaller generating sets for groups in composition/chief series, preimages, all subgroups
- Better permutation representations:

Assume $N \triangleleft G$ el.ab., U point stabilizer in permutation action of G that intersects into N. Then UN stabilizes submodule $U \cap N$. Use information about subgroups of G/N to get permutation representation for G.



Presentations

Composition tree uses presentations (of simple groups) to verify the tree.

GAP currently has canned presentations only for alternating groups, otherwise builds from stabilizer chain. ➠Larger, more arbitrary, slower

Could use:

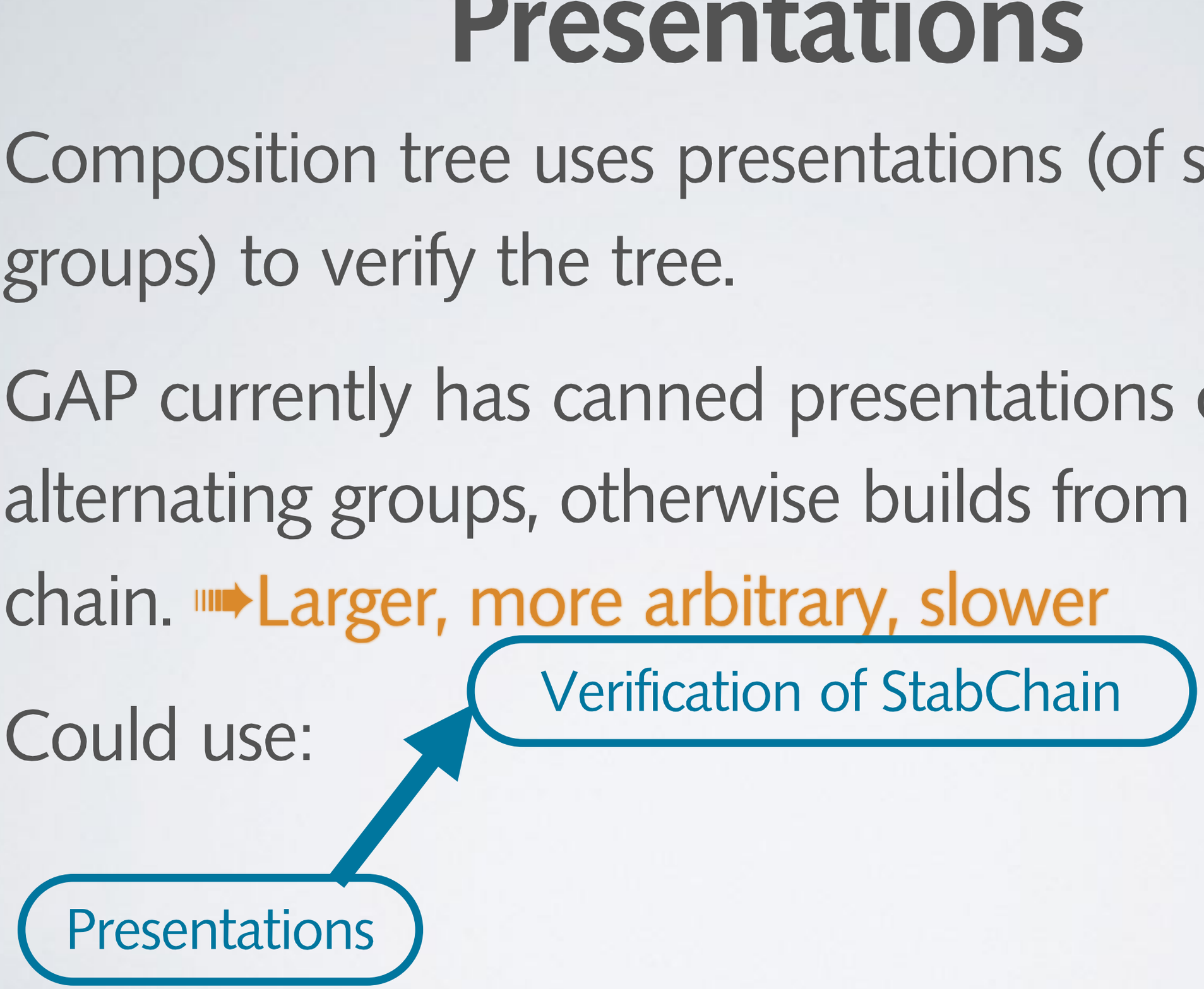
Presentations

Presentations

Composition tree uses presentations (of simple groups) to verify the tree.

GAP currently has canned presentations only for alternating groups, otherwise builds from stabilizer chain. ➠Larger, more arbitrary, slower

Could use:



Presentations

Composition tree uses presentations (of simple groups) to verify the tree.

GAP currently has canned presentation
alternating groups, otherwise builds from
chain. ➠Larger, more arbitrary, slower

currently: combinatorial verification

Could use:

Verification of StabChain

Presentations

Presentations

Composition tree uses presentations (of simple groups) to verify the tree.

GAP currently has canned presentations only for alternating groups, otherwise builds from stabilizer chain. ➠Larger, more arbitrary, slower

Could use:

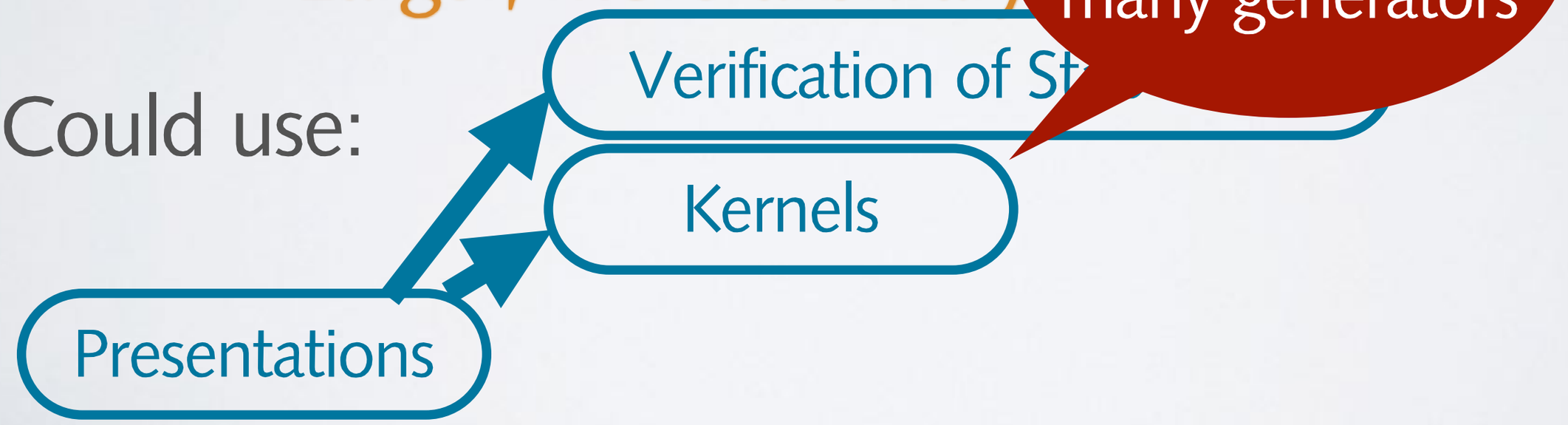
Presentations

Composition tree uses presentations (of simple groups) to verify the tree.

GAP currently has canned presentations only for alternating groups, otherwise b… bilizer chain. ➠Larger, more arbitrary…

currently: stabilizer chain, many generators

Could use:



Presentations

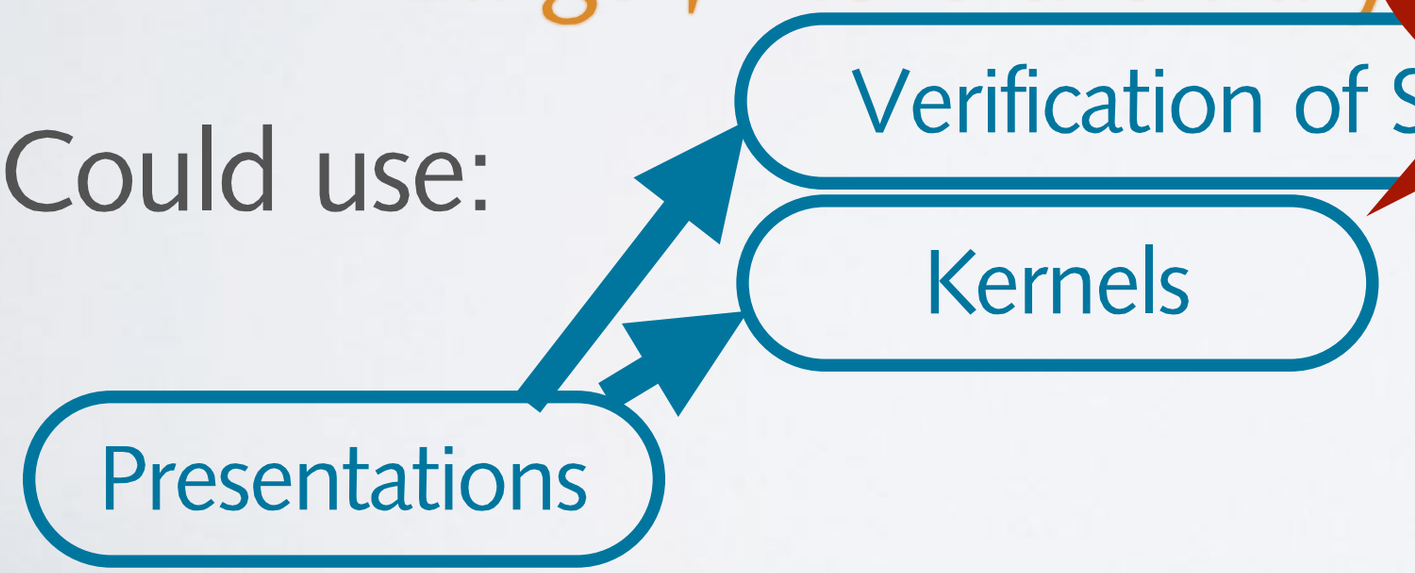
Composition tree uses presentations (of simple groups) to verify the tree.

GAP currently has canned presentations only for alternating groups, otherwise b… bilizer chain. ➠Larger, more arbitrary

currently: stabilizer chain, many generators

Better: Random elements

Could use:

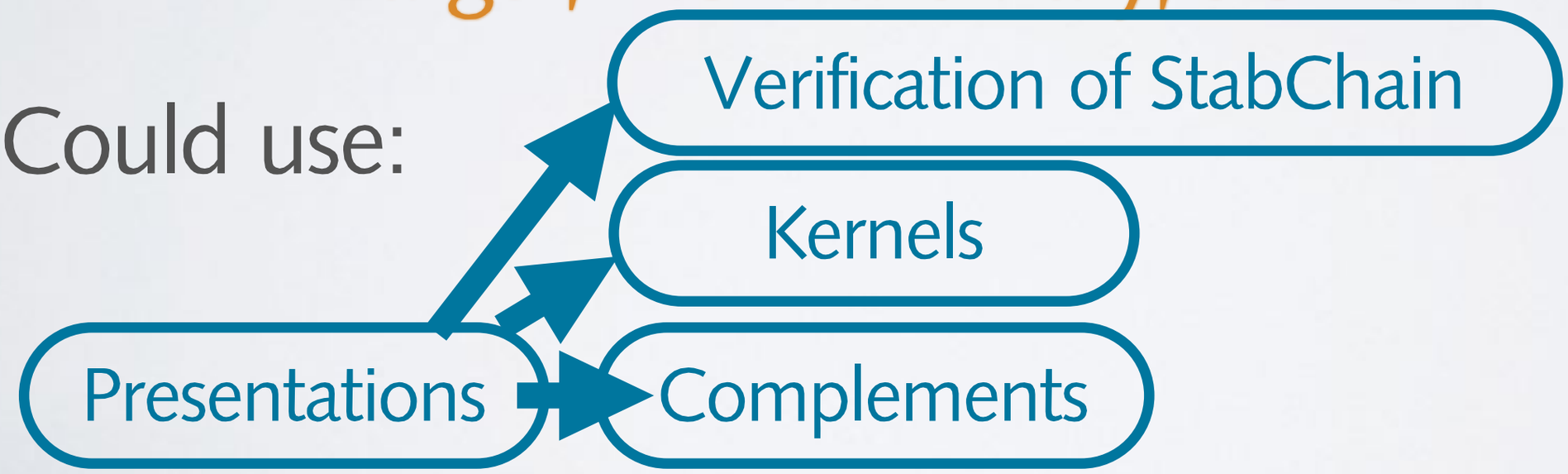


Presentations

Composition tree uses presentations (of simple groups) to verify the tree.

GAP currently has canned presentations only for alternating groups, otherwise builds from stabilizer chain. ➠Larger, more arbitrary, slower

Could use:

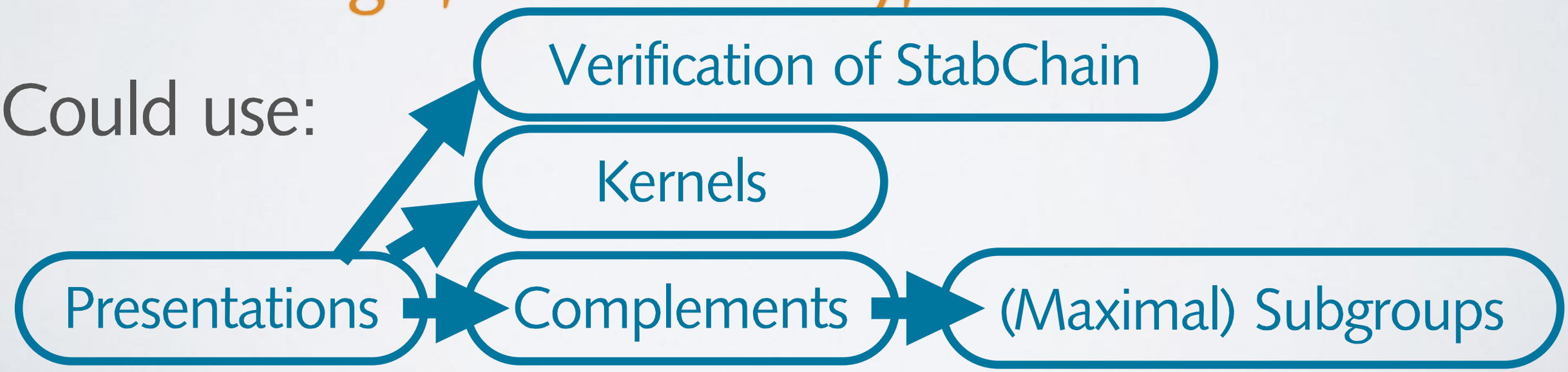


Presentations

Composition tree uses presentations (of simple groups) to verify the tree.

GAP currently has canned presentations only for alternating groups, otherwise builds from stabilizer chain. ➠Larger, more arbitrary, slower

Could use:



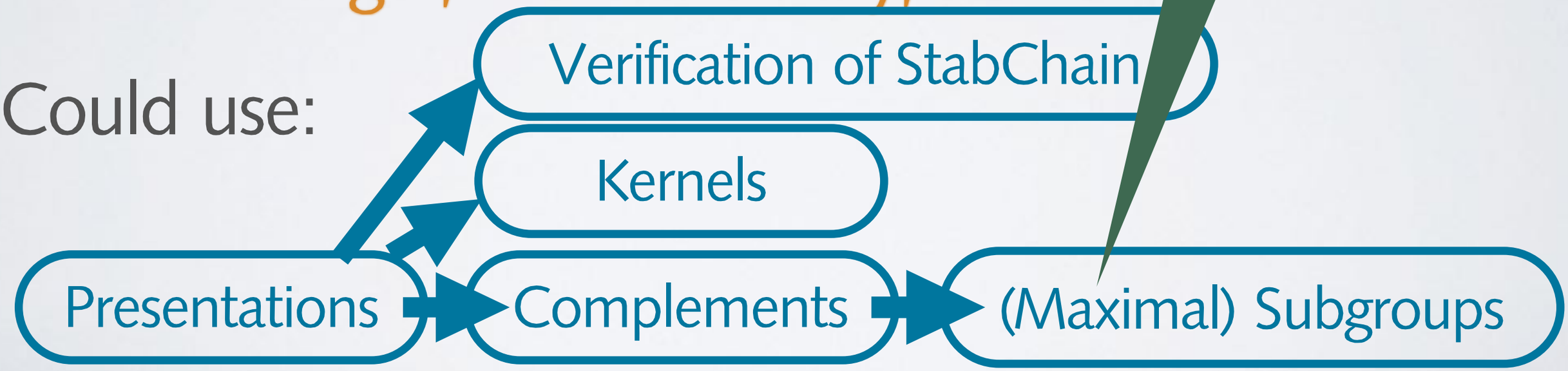
Presentations

Composition tree uses presentations (of simple groups) to verify the tree.

GAP currently has canned pres
alternating groups, otherwise bu
chain. ➠Larger, more arbitrary, slower

Should be
a cheap operation, cost
comparable to
composition series

Could use:

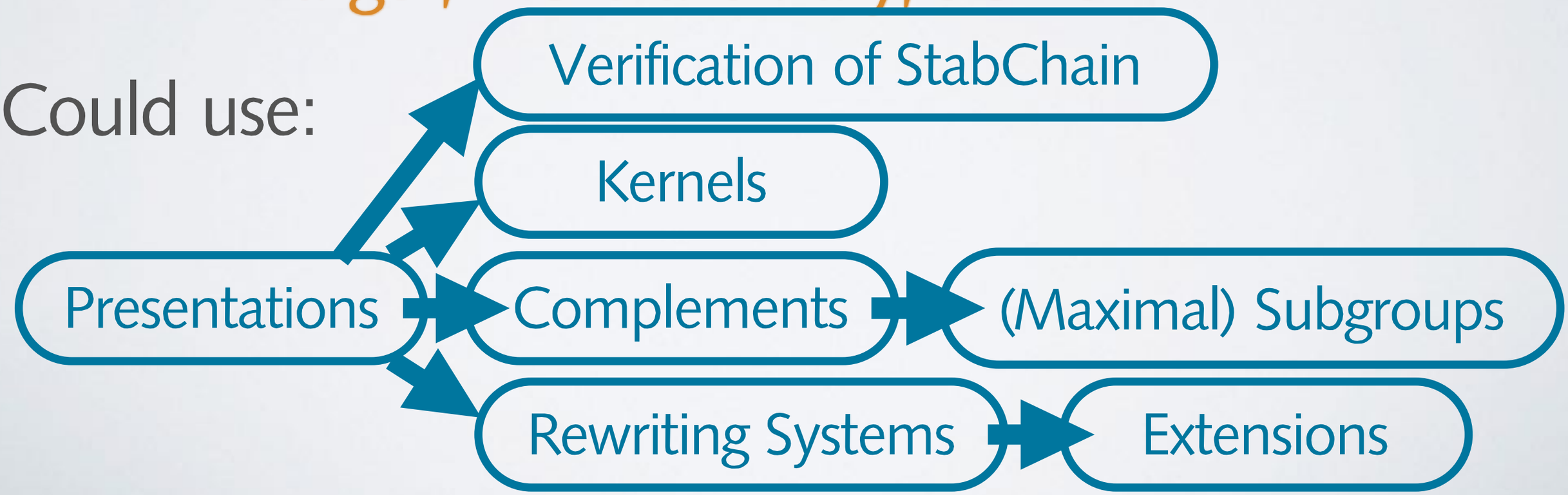


Presentations

Composition tree uses presentations (of simple groups) to verify the tree.

GAP currently has canned presentations only for alternating groups, otherwise builds from stabilizer chain. ➠Larger, more arbitrary, slower

Could use:



Verification Of Stabilizer Chains

Доверяй, но проверяй

Trust, but verify

```
gap> f:=FreeGroup("a","b","c");;
gap> g:=f/ParseRelators(f,
    "a^3=b^4=c^4=(a^2b^2)^2=(a^2c)^2=(abc)^2=1");;
gap> l:=LowIndexSubgroups(g,10);;Length(l);
47
gap> ProfileOperationsAndMethods(true);
gap> ProfileGlobalFunctions(true);
gap> k:=Intersection(l);;
gap> DisplayProfile();
  count  self/ms  chld/ms    function
     61     201    51119    VerifySGS
     99       9   104373    StabChainRandomPermGr*
   2143      16   142622    StabChainOp: group an*
     46       2   164512    Intersection2: subgro*
         164636             TOTAL
gap>
```

```
gap> f:=FreeGroup("a","b","c");;
gap> g:=f/ParseRelators(f,
    "a^3=b^4=c^4=(a^2b^2)^2=(a^2c)^2=(abc)^2=1");;
gap> l:=LowIndexSubgroups(g,11);;Length(l);
53
gap> ProfileOperationsAndMethods(true);
gap> ProfileGlobalFunctions(true);
gap> k:=Intersection(l);;
gap> DisplayProfile();
  count   self/ms   chld/ms    function
    117       820   1201184    VerifySGS
    179        41   1676427    StabChainRandomPermGr*
  29633       246   1996006    StabChainOp: group an*
     52         4   2271196    Intersection2: subgro*
          2271210               TOTAL
gap>
```

```
gap> f:=FreeGroup("a","b","c")
gap> g:=f/ParseRelators(f,
    "a^3=b^4=c^4=(a^2b^2)^
gap> l:=LowIndexSubgroups
53
gap> ProfileOperationsAnd
gap> ProfileGlobalFunctions
gap> k:=Intersection(l);;
gap> DisplayProfile();
  count   self/ms   chld/ms    function
    117       820   1201184    VerifySGS
    179        41   1676427    StabChainRandomPermGr*
  29633       246   1996006    StabChainOp: group an*
     52         4   2271196    Intersection2: subgro*
          2271210               TOTAL
gap>
```

>50% of calculation time is spent on verifying a stabilizer chain. **Testing takes longer than the actual chain computations**!

Verification Of Stabilizer Chains

The current verification can be very costly in cases such as groups with many orbits (as arise naturally when combining permutation quotients — `FpGroups` and `NaturalHomomorphismByNormalSubgroup`).

Analog case: Number of generators for permutation subgroups (backtrack, kernel, pre-image).

Presentation-based verification ought to be much faster, as the constituents are “easy”.

Verification

Use the existing `CompositionSeries` for permutation groups, or composition tree code?

The current verifica

such as groups with

when combining permutation quotients — `FpGroups` and `NaturalHomomorphismByNormalSubgroup`).

Analog case: Number of generators for permutation subgroups (backtrack, kernel, pre-image).

Presentation-based verification ought to be much faster, as the constituents are “easy”.

Verification

The current verifica

such as groups with

when combining pe

and `NaturalHomomo`

Analog case: Numb

subgroups (backtrack, kernel, pre-image).

Presentation-based verification ought to be much faster, as the constituents are “easy”.

Use the existing `CompositionSeries` for permutation groups, or composition tree code?

Does Verification need to give hints about omissions, or simply trigger recalculation?

Verification

The current verifica

such as groups with

when combining pe

and NaturalHomomo

Analog case: Numb

subgroups (backtrack, kernel, pre-image).

Presentation-based

faster, as the consti

Use the existing CompositionSeries for permutation groups, or composition tree code?

Does Verification need to give hints about omissions, or simply trigger recalculation?

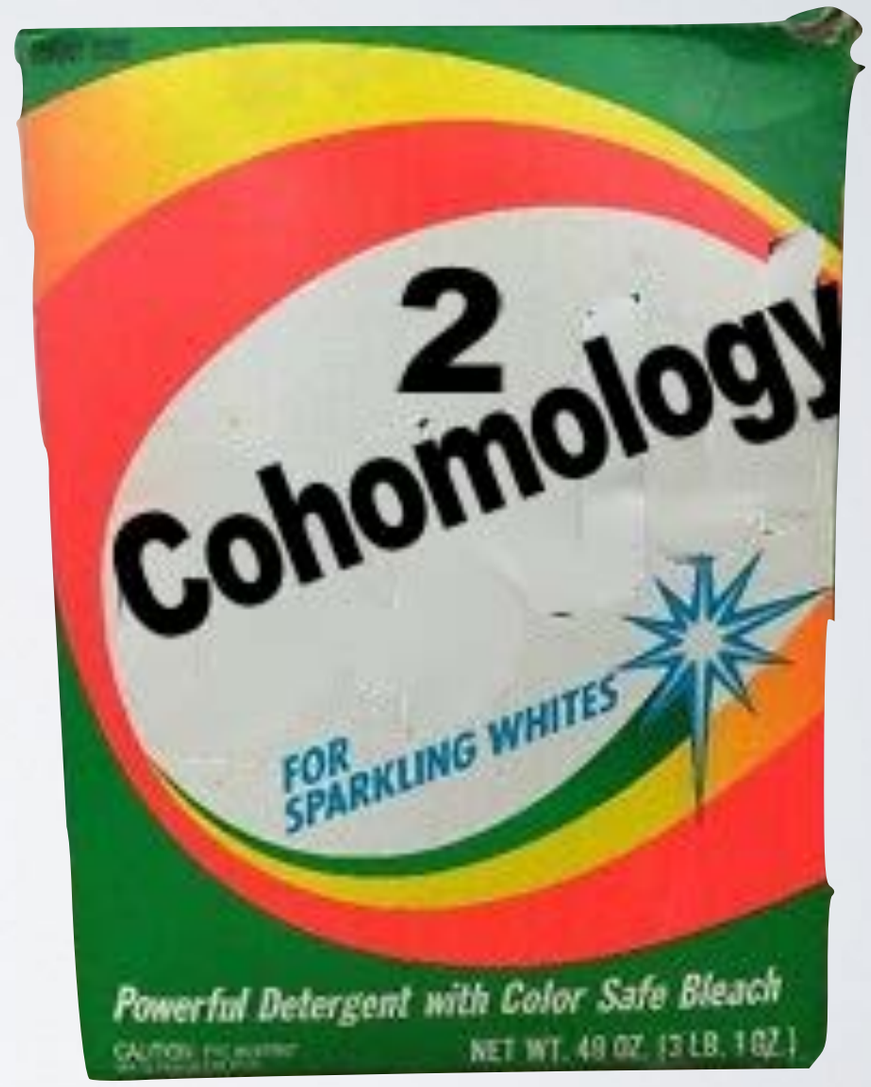
Also for genss package!

Advertisement: 2-Cohomology

Current development version has 2-Cohomology for general finite groups: `TwoCohomologyGeneric`.

Uses confluent rewriting system for factor group, pair overlaps to get conditions.

`FpGroupCocycle` constructs extension (also can build decent permutation representation.)



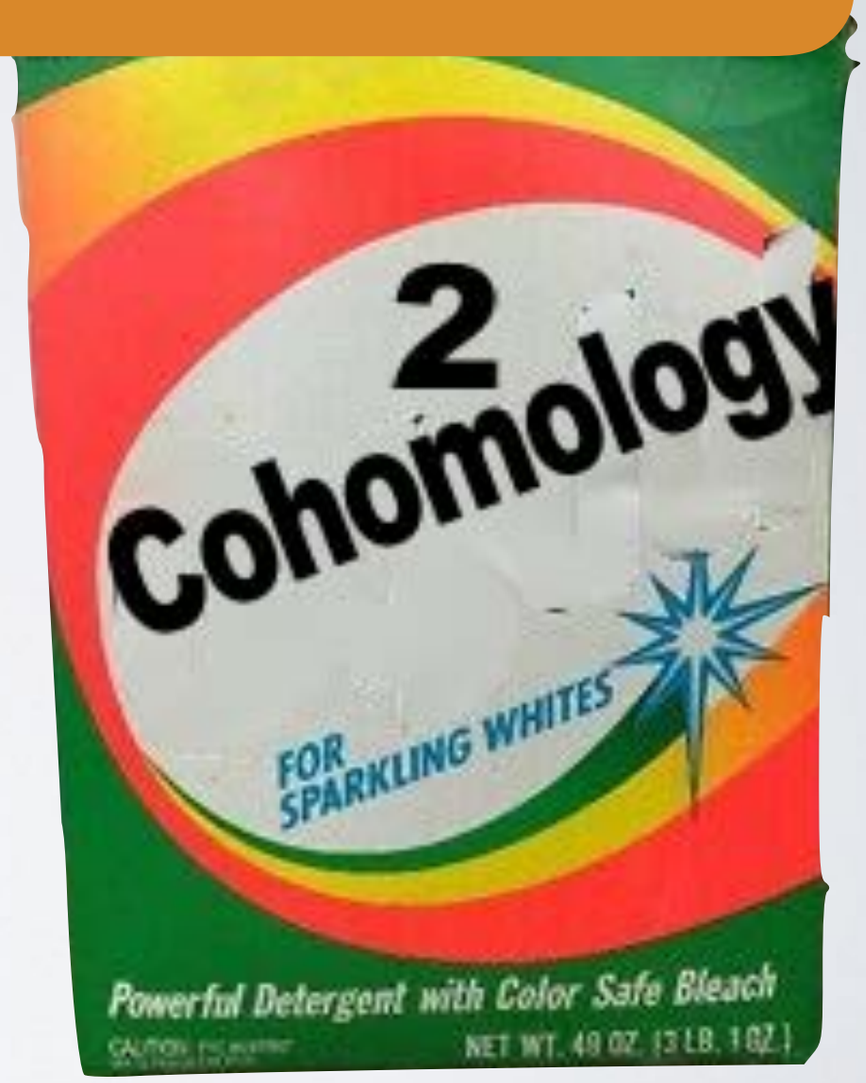
Advertisem

Current developmer
has 2-Cohomology
finite groups:
`TwoCohomologyGeneric`.

Uses confluent rewriting system for factor group, pair overlaps to get conditions.

`FpGroupCocycle` constructs extension (also can build decent permutation representation.)

! So far good rewriting systems for simples only for alternating groups.



```
gap> g:=PerfectGroup(IsPermGroup,1344,1);;
gap> mo:=IrreducibleModules(g,GF(2),0);;List(mo[2],x-
>x.dimension);
[ 1, 3, 3, 8 ]
gap> coh:=TwoCohomologyGeneric(g,mo[2][2]);;
gap> coh.cohomology;
[ <an immutable GF2 vector of length 159>,
  <an immutable GF2 vector of length 159> ]
gap> comp:=CompatiblePairs(g,mo[2][2]);
<group of size 2688 with 5 generators>
gap> reps:=CompatiblePairOrbitRepsGeneric(comp,coh);;
gap> Length(reps);
3
gap> gps:=List(reps,x->FpGroupCocycle(coh,x,true));;
gap> gps:=List(gps,x->Image(IsomorphismPermGroup(x)));
[ <perm. group with 8 generators>, <perm. group with 8
    generators>, <perm. group with 8 generators> ]
gap> List(gps,NrMovedPoints);
[ 16, 22, 28 ]
gap> List(gps,MinimalFaithfulPermutationDegree);
[ 16, 22, 28 ]
```

build permutations ⬇

Example: Perfect Groups

Currently testing by constructing perfect groups (also uses `CompatiblePairOrbitRepsGeneric`):

There are	98	**52**	258	154
perfect groups of order	61440	86016	122880	172032

There are	>500	291	**46**
perfect groups of order	245760	344064	368640

(and I have concrete lists of groups)

Recognition

To use presentations, need constructive recognition of (almost) simple groups. It has many other uses:

- Maximal subgroups (already needed, generic isomorphism routine requires conjugacy classes)
- **Could do:** Better permutation representation
- Conjugacy classes
- Particular/All subgroups
- Automorphism group (reps for outer)
- In some cases, a groups construction could provide recognition for free.

Recognition

To use presentations, need constructive recognition of (almost) simple groups. It has many other uses:

- Maximal subgroups (already needed, generic isomorphism routine requires conjugacy classes)
- **Could do:** Better permutation representation
- Conjugacy classe
- Particular/All su
- Automorphism
- In some cases, a provide recogni

> **!** Need Recognition operation in library (or always loaded package) with fallback based on isomorphism test.

Recognition

To use presentations, need constructive recognition of (almost) simple groups. It has many other uses:

- Maximal subgroups (already needed, generic isomorphism routine requires conjugacy classes)
- **Could do:** Better permutation representation
- Conjugacy classes
- Particular/All subgroups
- Automorphism group (reps for outer)
- In some cases, a groups construction could provide recognition for free.

Recognition

To use presentations, need constructive recognition of (almost) simple groups. It has many other uses:

- Maximal subgroups (already needed, generic isomorphism routine requires conjugacy classes)
- **Could do:** Bette
- Conjugacy clas
- Particular/All su
- Automorphism
- In some cases, provide recogn

Immediate pay-off: Maximal subgroups, and routines that use it (intermediate subgroups, **double cosets**, factor perm rep., automorphism group)

Recognition

To use presentations, need constructive recognition of (almost) simple groups. It has many other uses:

- Maximal subgroups (already needed, generic isomorphism routine requires conjugacy classes)
- **Could do:** Better permutation representation
- Conjugacy classes
- Particular/All subgroups
- Automorphism group (reps for outer)
- In some cases, a groups construction could provide recognition for free.

Recognition

To use presentatio
of (almost) simple

- Maximal subgrou
  isomorphism rou

> ! Isomorphism can not use recognition as long as recognition might fall back to isomorphism.

- **Could do:** Better permutation representation
- Conjugacy classes
- Particular/All subgroups
- Automorphism group (reps for outer)
- In some cases, a groups construction could provide recognition for free.

Proposed Recognition API

- Attribute `ConstructiveRecognition` returns type information and an object on which one can call `ImagesRepresentative` and `PreImagesRepresentative`.
- Operation `RecognizeGroup(type,group)` returns such homomorphism object for known type. The type is either a record, or `true` (if type is not known).
- Methods can dispatch on group, but otherwise always apply, bail out if type does not fit.
- Fallback method: Calculate order, run through simple groups of that order, test isomorphism.

Proposed Recognition API

- Attribute `ConstructiveRecognition` returns type information and an object on which one can call
  `esRepresentative`.
  `oup`) returns such
  type. The type is
  not known).
  but otherwise
  not fit.
  , run through
  somorphism.

Type is record with components series and parameter (which could be string for sporadic). **Must be fixed once and for all**. Build on current `DataAboutSimpleGroup` and `SimpleGroup`

Proposed

- Attribute Construc
information and a

Need translation from type to name used by character tables, tables of marks.

esRepresentative.

oup) returns such

type. The type is

not known).

but otherwise

not fit.

, run through

somorphism.

Type is record with components series and parameter (which could be string for sporadic). **Must be fixed once and for all**. Build on current DataAboutSimpleGroup and SimpleGroup

Proposed Recognition API

- Attribute `ConstructiveRecognition` returns type information and an object on which one can call `ImagesRepresentative` and `PreImagesRepresentative`.
- Operation `RecognizeGroup(type,group)` returns such homomorphism object for known type. The type is either a record, or `true` (if type is not known).
- Methods can dispatch on group, but otherwise always apply, bail out if type does not fit.
- Fallback method: Calculate order, run through simple groups of that order, test isomorphism.

Proposed Recognition API

- Attribute `ConstructiveRecognition` returns type information and an object on which one can call `ImagesRepresentative` and `PreImagesRepresentative`.
- Operation `RecognizeGroup(type,group)` returns such type. The type is not known).

Could be GAP homomorphism or less. No membership test. (Provide different interface if failure may be possible.)

but otherwise not fit.

, run through simple groups of that order, test isomorphism.

Proposed Recognition API

- Attribute `ConstructiveRecognition` returns type information and an object on which one can call `ImagesRepresentative` and `PreImagesRepresentative`.
- Operation `RecognizeGroup(type,group)` returns such homomorphism object for known type. The type is either a record, or `true` (if type is not known).
- Methods can dispatch on group, but otherwise always apply, bail out if type does not fit.
- Fallback method: Calculate order, run through simple groups of that order, test isomorphism.

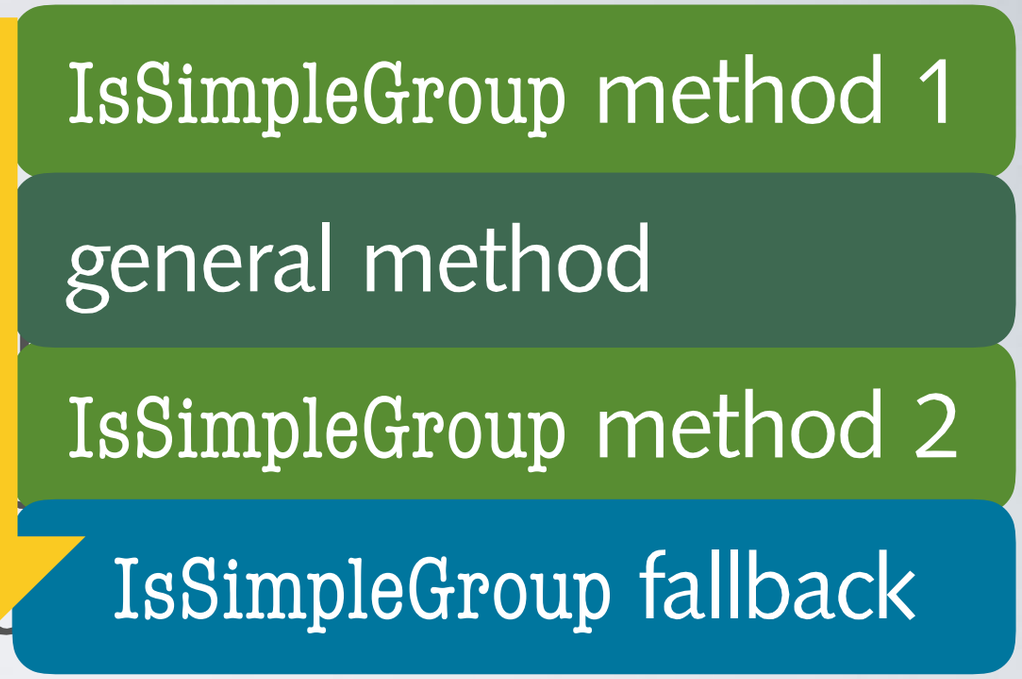
Proposal: Transfer Information

Any method for generic operation that reduces to simple group case will call appropriate/same operation (e.g. `MaximalSubgroupClassReps`) on argument for which `IsSimpleGroup` is set to true.

Convention: Such methods will `TryNextMethod()` if group is known simple, redispatch if simple discovered. Separate fallback method in library at `IsSimpleGroup` rank only, using old approach (used so far).

Libraries that provide data install methods for operation under (at least) `IsSimpleGroup`. (installed later, so rank higher than library fallback).

Proposal: Transfer



Any method for generic operatic
simple group case will call appro
(e.g. MaximalSubgroupClassReps)
which IsSimpleGroup is set to true.

Convention: Such methods will TryNextMethod() if group is known simple, redispatch if simple discovered. Separate fallback method in library at IsSimpleGroup rank only, using old approach (used so far).

Libraries that provide data install methods for operation under (at least) IsSimpleGroup. (installed later, so rank higher than library fallback).

Other Composition Trees

Same kind of data structure for other classes of groups. Can use some actions and permutation fallback in place of Aschbacher theorem:

- Automorphism groups (action on group layers) [cf. Sims '97]
- Hybrid groups (formal extensions), formal factor groups: Use composition tree interface to get high-level algorithms.
- Permutation groups (e.g. if not short-base).

Summary: Agenda

1. Provide short presentations for simple groups, enable recognition verification.
2. Define Constructive recognition API. Implement methods for A_n, PSL_n (at least small n).
3. Use presentations to verify stabilizer chains
4. Use presentations to verify composition trees
5. Allow separation of `HasNice...` methods. Model: methods for generic operations for `FpGroups`.
6. Start using composition tree for matrix groups.
7. Start looking at other groups (automorphisms &c.)