1 July 2019

The `recog` package

Max Horn
University of Siegen

WARNING!

This talk contains very little math

Some background

What is `recog` and why does it matter?

- `recog` is a `GAP` package
- `recog` implements {matrix, permutation, black box}-group recognition
- many computational research problems hinge on having access to effective group recognition
- only (?) matrix group recognition implementation outside of `Magma`
- fully operational `recog` is crucial for `GAP` and beyond

What happened so far …

- `recog` authors: Ákos Seress and Max Neunhöffer
- 2013 was a bad year: Ákos died; Max left academia
- since then, no major work was done on `recog`
- (some minor work by me: bug fixes, documentation, …)
- lots is missing in `recog` – but we need it!
- who is going to do the work?

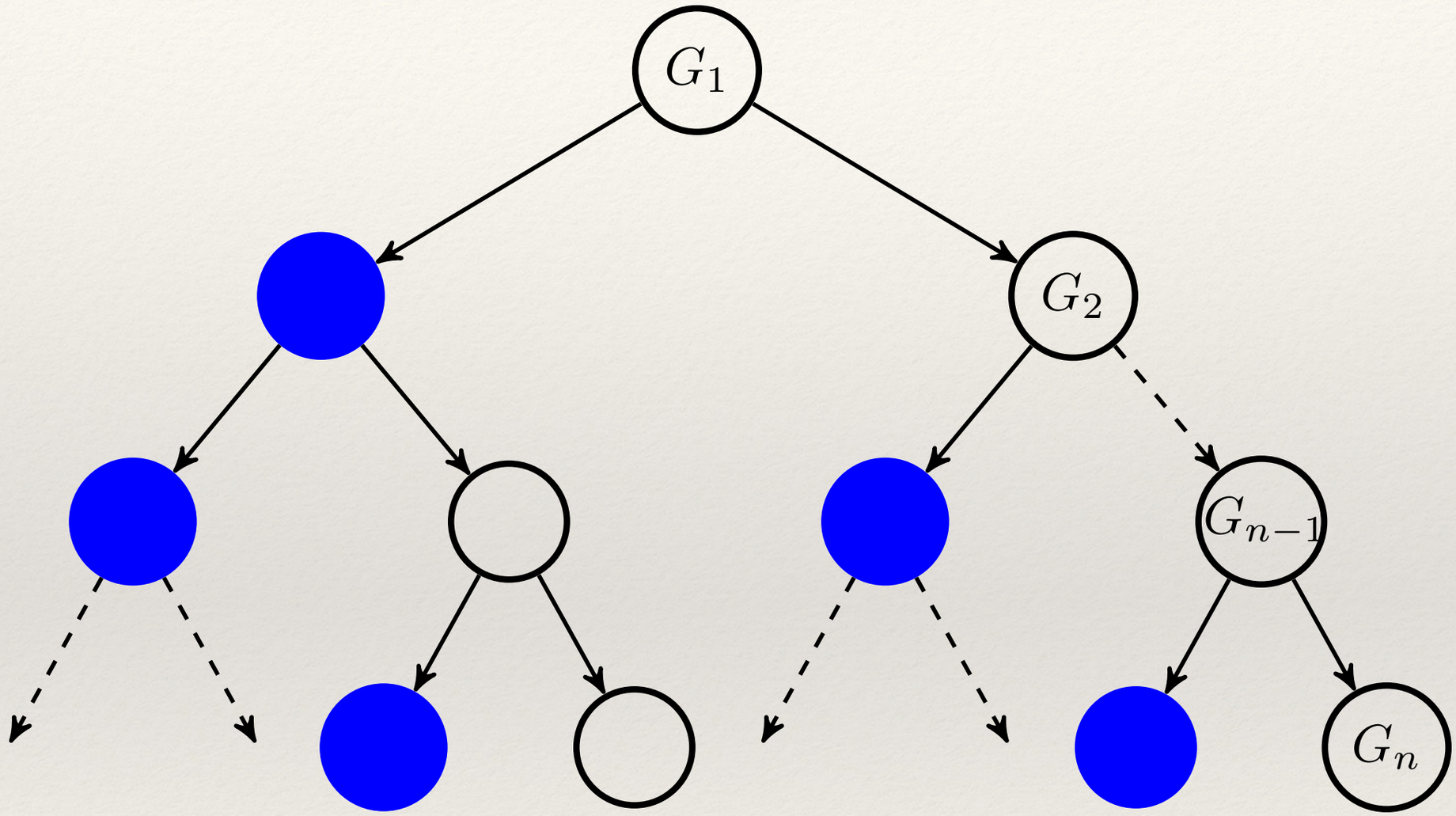
… and your students, friends, … – spread the word!

How does it work?

How does recognition work, roughly?

- *Input:* set X of generators (permutations, matrices, …)
- *Output:*
  - order of $G := \langle X \rangle$, name of G
  - new generating set Y,
  - procedure to express any $g \in G$ as word (SLP) in Y
- Iterate over a catalog of methods to analyze G; each either
  - "reduces" to smaller cases by an epimorphism $\varphi : G \to H$, or
  - handles group directly if "easy" or almost simple

The recognition tree



- blue = kernels, white = images
- leaves are either easy to handle or almost simple

Writing recognition methods

What is a recognition method?

- we start with input $G = \langle X \rangle$, and reduce to subquotients
- now we study the subquotient $K = \langle X_K \rangle$
- a *recognition method* is a procedure with two tasks:

  (1) find new "nice generators" Y_K for K together with a procedure which expresses any $k \in K$ as SLP in Y_K

  (2) record how Y_K was derived from X_K (e.g. via SLPs)

Method selection and hints

- `recog` has "database" of recognition methods
- methods are ordered by a "rank"
- `recog` tries methods in order of rank, until one succeeds
- methods may be retried depending on their return value
- methods may pass "hints" to factor and kernel, and also specify additional recognition method

Methods on a technical level

- Input: recognition info record `ri` and a group `G`
- when run, attempt to resolve the *two tasks* somehow
- if successful:
  - update `ri` with new generators Y_K; function for producing SLP in Y_K for any $k \in K$; information how to derive Y_K from X_K (e.g. SLPs)
  - return `Success`
- else return one of …
  - `NeverApplicable` or `TemporaryFailure` or `NotEnoughInformation`

A trivial example

```
SLPforElementFuncs.TrivialGroup := function(ri, g)
  return StraightLineProgramNC( [ [1,0] ], 1 );
end;

FindHomMethodsGeneric.TrivialGroup := function(ri, G)
  local gens;
  gens := GeneratorsOfGroup(G);
  if not ForAll(gens, ri!.isone) then
      return NeverApplicable;
  fi;
  SetSize(ri, 1);
  Setslpforelement(ri, SLPforElementFuncs.TrivialGroup);
  Setslptonice(ri, StraightLineProgramNC([[[1,0]]],
                   Length(gens)));
  SetFilterObj(ri, IsLeaf);
  return Success;
end;

AddMethod(FindHomDbPerm,
          FindHomMethodsGeneric.TrivialGroup,
          300, "TrivialGroup",
          "go through generators, compare to identity");
```

A branching example

- Any example that is not a leaf node involves a nontrivial epimorphism $\varphi : K \to H$
- define nice gens of K as union of nice gens Y_N of $N := \ker \varphi$, and preimages of nice gens Y_H of H
- want SLP in Y_K for $k \in K$:
  - express $\varphi(k)$ as SLP in Y_H
  - evaluate this over Y_K: yields $k' \in Nk$
  - express kk'^{-1} as SLP in Y_N
  - combined we get SLP for k in Y_K

```
FindHomMethodsPerm.NonTransitive :=
  function(ri, G)
    local hom, la, o;

    # Then test whether we can do something:
    if IsTransitive(G) then
        return NeverApplicable;
    fi;

    la := LargestMovedPoint(G);
    o := Orb(G, la, OnPoints);
    Enumerate(o);
    hom := OrbActionHomomorphism(G, o);
    SetHomom(ri, hom);
    return Success;
  end;

AddMethod(FindHomDbPerm,
          FindHomMethodsPerm.NonTransitive,
          90, "NonTransitive",
          "try to restrict to orbit");
```

Passing hints to image and kernel

```
FindHomMethodsMatrix.BlockLowerTriangular := function(ri, G)
  # This is only used coming from a hint, we know what to do:
  # A base change was done to get block lower triangular shape.
  # We first do the diagonal blocks, then the lower p-part:
  local H, data, hom, newgens;
  data := rec( blocks := ri!.blocks );
  newgens := List(GeneratorsOfGroup(G), x -> RECOG.HomOntoBlockDiagonal(data, x));
  Assert(0, not fail in newgens);
  H := Group(newgens);
  hom := GroupHomByFuncWithData(G, H, RECOG.HomOntoBlockDiagonal, data);
  SetHomom(ri, hom);

  # Give hint to factor
  forfactor(ri).blocks := ri!.blocks;
  Add(forfactor(ri).hints, rec( method := FindHomMethodsMatrix.BlockDiagonal,
                                rank := 2000, stamp := "BlockDiagonal" ) );
  # Give hint to kernel N
  findgensNmeth(ri).method := FindKernelLowerLeftPGroup;
  findgensNmeth(ri).args := [];
  Add(forkernel(ri).hints, rec( method := FindHomMethodsMatrix.LowerLeftPGroup,
                                rank := 2000, stamp := “LowerLeftPGroup" ));
  forkernel(ri).blocks := ri!.blocks;

  return Success;
end;
```

What is there and what is missing?

What is there?

- a flexible framework for tying together different recognition methods
- various recognition methods are already implemented
- list in the `recog` manual
- https://gap-packages.github.io/recog/doc/chap6.html

What is missing?

- Many old and new methods for recognizing almost simple groups were never implemented
- most constructive recognition methods
- Verification is not implemented — very important!
  - need to add presentations to “leaf” nodes / recognition methods
  - then provide infrastructure to lift these through the recognition tree

What else is missing?

- Higher-level methods that rewrite or use recognition tree (see Eamonn's talk)
- Bug fixes
- More and better documentation
- Tests, tests, tests (import from Magma)
- Performance improvements
- Infrastructure on the `GAP` side: faster MeatAxe, `MatrixObj` (new matrix interface), …

Help wanted!

- Anybody who is interested in having group recognition in GAP: please consider contributing to `recog`
- Some ways to help are listed on the summer school website, see also https://bit.ly/recog-tasks

Developer infrastructure

- Homepage
  - https://gap-packages.github.io/recog/
- Source code
  - https://github.com/gap-packages/recog/
- Issue tracker (bug reports, feature requests, support)
  - https://github.com/gap-packages/recog/issues/
- Continuous integration / tests
  - https://travis-ci.org/gap-packages/recog/

References

- Max Neunhöffer, Ákos Seress, *A data structure for a uniform approach to computations with finite groups*, 2006
- Max Neunhöffer, *Constructive Recognition of Finite Groups*, 2009, habilitation thesis
- Henrik Bäärnhielm, Derek Holt, Charles Leedham-Green, Eamonn O'Brien, *A practical model for computation with matrix groups*, 2014