AN INTRODUCTION TO GAP

Steve Linton
Division of Computer Science, St. Andrews

July 7, 1997

Overview

- Part I
  - Computation and algebra – generalities
  - Getting Started with GAP
  - Simple Calculations with numbers and groups
  - Getting help, reading files, logging and saving
  - Lists and records – useful functions for lists
- Part II
  - Conditionals and Loops
  - Writing functions
  - A little about Soluble Groups in GAP

GENERALITIES

Why Compute?

- Exploration – to form or dismiss conjectures or gain insight
- Resolve finite problems – sporadic structures, small cases left over by general theorems
- Teaching – allow students to work out non-trivial examples
- As a research area in its own right – algorithms, algorithmics
- As a testbed for Computer Science – type systems, languages, database concepts

Approaching Problems in Computational Algebra

- Charlie Sims, some years ago, divided problems into three classes

  I Problems where many group elements fit into memory

  II Problems where just a few group elements fit

  III Problems where only a fraction of one element fits

- I would generalise this and speak of:

  I Problems that can be solved quickly by entirely automatic means

  II Problems that can be solved using standard low-level tools and some mathematical ingenuity

  III Problems that require special-purpose programming

- These boundaries are both fuzzy and moving
- Frustration arises when you tackle a problem in one class using tools suitable for another

The Reds and the Greens

- Two schools in CGT (especially pre 1991)
- Green school (Sims, Europeans, Australians, mainly mathematicians)
  - Pragmatists
  - Implementation is important
  - "Good" algorithm performs well on standard or big examples
- Red school (North America, mainly computer scientists)
  - Complexity theorists – started from Graph Isomorphism problem
  - Precise complexity calculation important
  - "Good" algorithm has low asymptotic complexity
- Some fusion since 1991 – Seress, Cooperman
- Result – better theory and better implementations

GETTING STARTED WITH GAP

GAP

- Groups, Algorithms, Programming
- Free software
- Runs under UNIX, MSDOS and MacOS (at least) – easy to port
- Developed over 11 years in Lehrstuhl D für Mathematik, RWTH Aachen, Germany
  - Prof Joachim Neubüser
  - Martin Schönert
  - Cast of thousands, mainly in Aachen, some elsewhere
- Headquarters moving to St Andrews in the next few weeks
- Current version 3.4.4. Version 4 about to enter β-test
- More info `http://www-gap.dcs.st-and.ac.uk/~gap/`
- Download the system from `ftp://ftp-gap.dcs.st-and.ac.uk/pub/gap`

First Steps

```
{caolila:106} gap -b

                 ########            Lehrstuhl D fuer Mathematik
               ###    ####           RWTH Aachen
              ##         ##
             ##          #             #######            #########
            ##                        #      ##          ## #     ##
            ##           #           #       ##             #      ##
            ####        ##           ##       #             #      ##
             #####     ###           ##      ##             ##    ##
               ######### #            #########             #######
                         #                                  #
                        ##           Version 3              #
                       ###           Release 4.4            #
                      ## #           18 Apr 97              #
                     ##  #
                    ##   #  Alice Niemeyer, Werner Nickel,  Martin Schoenert
                   ##    #  Johannes Meier, Alex Wegner,    Thomas Bischops
                  ##     #  Frank Celler,   Juergen Mnich,  Udo Polis
                  ###   ##  Thomas Breuer,  Goetz Pfeiffer, Hans U. Besche
                   ######   Volkmar Felsch, Heiko Theissen, Alexander Hulpke
                            Ansgar Kaup,    Akos Seress,    Erzsebet Horvath
                            Bettina Eick
                            For help enter: ?<return>
gap>
```


The GAP Prompt

- What GAP displays `gap>` it is waiting for you to tell it what to do
- If you type an expression, followed by a `;` GAP will print its value:

```
gap> 1+1;
2
gap> 2^70;
1180591620717411303424
gap> Factorial(11);
39916800
gap>
```

- As well as integers, GAP can compute with other things

```
gap> (1,2,3,4)^2;
(1,3)(2,4)
gap> Size(GF(5));
5
gap> Elements(SymmetricGroup(3));
[ (), (2,3), (1,2), (1,2,3), (1,3,2), (1,3) ]
gap> Z(4)^2 = Z(2);
false
gap> Z(4)^3 = Z(2);
true
gap> true and false;
false
gap>
```

Variables

- To remember computed values for later, you can assign them to variables

```
gap> a := 2*3;
6
gap> a+1;
7
gap> b := a;
6
gap> a := a+1;
7
gap> a; b;
7
6
```

- `last` is a special variable (and `last2` and `last3`)

```
gap> 5/7;
5/7
gap> last;
5/7
gap> 7/5; last;
7/5
7/5
gap> last+1;
12/5
```

- It is best always to use lower-case for your variables

Supressing Printing

- If you are assigning a complicated value to a variable, you may not want to print it
- Using `;;` at the end of the line prevents printing

```
gap> e  := Elements(SymmetricGroup(4));;
gap> Length(e);
24
gap> Number(e,p->OrderPerm(p) = 3);
8
```

- Just using the built-in functions and assignment, you can do a lot of calculations

HELP, READING FILES, SAVING

Getting Help

- The GAP manual (all 1600 pages) can be read in four ways:
  - Via GAP itself, as on-line help
  - As a `dvi` or `ps` file, with a previewer or on paper
  - From the HTML files in the distribution, using a Web browser
  - On one of the GAP Web sites – St Andrews, Aachen, Northeastern U or ANU
- The On-line help is accessed using `?`

```
gap> ?
    Welcome to GAP ___________________________________________ Welcome to GAP

    Welcome to GAP.

    GAP is a system for computational group theory.

    Enter ´?About GAP´    for a step by step introduction to GAP.
    Enter ´?Help´         for information how to use the GAP help system.
    Enter ´?Chapters´     for a list of the chapters of the GAP help system.
    Enter ´?Copyright´    for the terms under which you can use and copy GAP.

    In each case do *not* enter the single quotes(´) , they are  used in help
    sections only to delimit text that you actually enter.

gap>
```

On-line help continued

- You can search the chapter and section headings using `??`

```
gap> ??dihedral
    dihedral _______________________________ Index
    The Basic Groups Library (DihedralGroup)
    TomDihedral
```

- The St Andrews Web pages provide a search facility for the whole text
- Note also `?>` to see the next section

Reading from Files

- When you do more complicated calculations, you want to be able to develop, debug and reuse them, without having to remember or retype what you did before
- You can create a file (usually called *file*`.g`) containing GAP commands and then `Read("`*file*`.g");` will cause GAP to execute those commands
- The results of commands in the file are *not* printed

```
{caolila:114} cat > test.g
1+3;
a := 17;
Print(a^3,"\n");
b := 11;
^D
{caolila:115} gap -b
gap> Read("test.g");
4913
gap> a;
17
gap> b;
11
```

- Note the `Print` command to directly print something

Logging

- If you are “exploring” some problem interactively, but want a record in case you make a discovery, you can use “logging”

```
gap> LogTo("test.log");
gap> s := SymmetricGroup(AgWords,4);
Group( g1, g2, g3, g4 )
gap> ccs := ConjugacyClasses(s);
[ ConjugacyClass( Group( g1, g2, g3, g4 ), IdAgWord ),
  ConjugacyClass( Group( g1, g2, g3, g4 ), g4 ),
  ConjugacyClass( Group( g1, g2, g3, g4 ), g2 ),
  ConjugacyClass( Group( g1, g2, g3, g4 ), g1 ),
  ConjugacyClass( Group( g1, g2, g3, g4 ), g1*g3 ) ]
gap> List(ccs,c -> Order(s,c.representative));
[ 1, 2, 3, 2, 4 ]
gap>
{caolila:117} cat test.log
gap> s := SymmetricGroup(AgWords,4);
Group( g1, g2, g3, g4 )
gap> ccs := ConjugacyClasses(s);
[ ConjugacyClass( Group( g1, g2, g3, g4 ), IdAgWord ),
  ConjugacyClass( Group( g1, g2, g3, g4 ), g4 ),
  ConjugacyClass( Group( g1, g2, g3, g4 ), g2 ),
  ConjugacyClass( Group( g1, g2, g3, g4 ), g1 ),
  ConjugacyClass( Group( g1, g2, g3, g4 ), g1*g3 ) ]
gap> List(ccs,c -> Order(s,c.representative));
[ 1, 2, 3, 2, 4 ]
```

- If you believe you have proved something by calculation, it is good practice to try and prepare a file which repeats the proof in as short and direct a way as possible. A log file is a good place to start from.

Saving Single Objects

- Sometimes you just want to store an object that you have computed.
- This can usually be done using the `PrintTo` command

```
gap> s := SymmetricGroup(10);
Group( ( 1,10), ( 2,10), ( 3,10), ( 4,10), ( 5,10), ( 6,10), ( 7,10),
( 8,10), ( 9,10) )
gap> c := Normalizer(s,Subgroup(s,[(1,2,3)(4,5,6,7,8)(9,10)]));
Subgroup( Group( ( 1,10), ( 2,10), ( 3,10), ( 4,10), ( 5,10), ( 6,10),
( 7,10), ( 8,10), ( 9,10) ), [ ( 1, 2, 3)( 4, 5, 6, 7, 8)( 9,10),
  ( 4, 6, 8, 5, 7), ( 1, 2, 3)( 4, 7, 5, 8, 6), ( 5, 6, 8, 7), ( 2, 3) ] )
gap> Size(last);
240
gap> PrintTo("cent.g","c := ",c,";\n");
gap> Unbind(c);
gap> Read("cent.g");
gap> Size(c);
240
```

- Note also `Unbind`. There is also `IsBound`.
- Not every object prints in a form that can be read (AG groups don’t)

Command Line Editing

- Can use `emacs` like keys to edit the command you are typing
- Ctrl-B to move back, Ctrl-F forward, Ctrl-P to get the previous line again, etc.
- See `?Line Editing`

LISTS AND RECORDS

Lists

- Data can be combined in GAP in two ways
- To talk about a lot of different things, we usually use a list
- For example `ConjugacyClasses` returns a list of classes
- We manipulate a list using `[ ]`.

```
gap> l := [2,3,5];
[ 2, 3, 5 ]
gap> l[1];
2
gap> l[3];
5
gap> l[7];
Error, List Element: <list>[7] must have a value
gap> l[7] := 17;
17
gap> l;
[ 2, 3, 5,,,, 17 ]
gap> Unbind(l[3]);
gap> l;
[ 2, 3,,,,, 17 ]
```


Records

- To talk about different aspects of the same thing, it is often better to use a record – groups are actually stored as records
- Manipulate using `rec` and `.`

```
gap> r := rec(age := 17, name := "fred", parents := ["bill
rec(
  age := 17,
  name := "fred",
  parents := [ "bill", "mary" ] )
gap> r.age;
17
gap> r.name;
"fred"
gap> r.pets;
Error, Record: element ´pets´ must have an assigned value
gap> r.pets := ["rover"];
[ "rover" ]
gap> Unbind(r.age);
gap> r;
rec(
  name := "fred",
  parents := [ "bill", "mary" ],
  pets := [ "rover" ] )
```

Useful Functions for Lists

- There are lots of useful functions for manipulating lists
  - `Length(l)` – last bound position
  - `Number(l)` – number of bound positions
  - `el in l` – is an element in the list
  - `Position(l,el)` – returns `false` is `l` is not there

```
gap> l;
[ 2, 3,,,,, 17 ]
gap> 3 in l;
true
gap> 5 in l;
false
gap> Length(l);
7
gap> Number(l);
3
gap> Position(l,3);
2
gap> Position(l,5);
false
```


Functions for Lists 2

- Also functions that apply other functions to lists
  - `List(l,func)` – The list of images
  - `ForAll(l,bfunc)` – does `bfunc` return true every time
  - `ForAny(l,bfunc)` – or any time
  - `Number(l,bfunc)` – how many times?
  - `First(l,bfunc)` – return the first element where it does
  - `Filtered(l,bfunc)` – make the list of all the places where it does

```
gap> l := [1..5];
[ 1 .. 5 ]
gap> List(l,x->CyclicGroup(x));
[ Group( () ), Group( (1,2) ), Group( (1,2,3) ), Group( (1
  Group( (1,2,3,4,5) ) ]
gap> ForAll(l,IsPrimeInt);
false
gap> ForAny(l,IsPrimeInt);
true
gap> First(l,IsPrimeInt);
2
gap> Filtered(l,IsPrimeInt);
[ 2, 3, 5 ]
gap> Number(l,IsPrimeInt);
3
```

- `[1..5]` is a special kind of list called a range. Also `[1,3..11]`
- See also `Sum`, `Product`, `Collected`

Identical Lists

- Two variables may refer to the *same* list
- Then changes to it are visible by either name
- To make a different list (with the same elements), use `ShallowCopy`

```
gap> l := [1,2,4];;
gap> m := l;;
gap> l[2] := 7;;
gap> m;
[ 1, 7, 4 ]
gap> m := ShallowCopy(l);
[ 1, 7, 4 ]
gap> l[2] := 11;;
gap> m;
[ 1, 7, 4 ]
```

PROGRAMMING

Statements

- As well as giving GAP expressions to evaluate, one can instruct it to do various things
- We have seen some examples of this already
  - assignments
  - system commands like `LogTo`
  - `Unbind`
- Sometimes these print a value, sometimes not
- One can also give conditional instructions

```
gap> g := OnePrimitiveGroup(DegreeOperation,50);
PSL(2,49)
gap> if IsSimple(g) then Print("simple\n");
> elif IsNilpotent(g) then print("nilpotent\n");
> else Print(List(CompositionSeries(g),Size));
> fi;
simple
```

- Must have `if`, `then` and `fi`, `elif` and `else` optional
- Can combine conditions with `and`, `or` and `not`

Loops

- The most common is the `for` loop

```
gap> for i in [1..10] do
> if IsPrime(i) then Print(i," "); else Print(i^2," "); fi
> od; Print("\n");
1 2 3 16 5 36 7 64 81 100
```

- You can use this to loop over any list:

```
gap> s := SymmetricGroup(5);; pi := ();;
gap> for class in ConjugacyClasses(s) do
> pi := pi*class.representative; od;
gap> pi;
(1,5,2,3)
```

- There are also `while` and `repeat` loops

Functions

- Although they can be useful on the command line, the main purpose of all these constructions is to write functions
- An example function:

```
gap> repeats := function(x,n)
> local l,i;
> l := [];
> for i in [1..n] do
> Add(l,x);
> od;
> return l;
> end;
function ( x, n ) ... end
```

- A function is an object like any other. You can assign it, pass it to a function, or return it from a function.

```
gap> copies := repeats;;
gap> copies(5,5);
[ 5, 5, 5, 5, 5 ]
gap> addx := function(x) return y->x+y; end;
function ( x ) ... end
gap> add3 := addx(3);
function ( y ) ... end
gap> add3(4);
7
gap> List([1..3],add3);
[ 4, 5, 6 ]
gap> l := List([1..3],addx);
[ function ( y ) ... end, function ( y ) ... end, function
gap> List(l,f -> f(5));
[ 6, 7, 8 ]
```

- Note the `->` shorthand for simple functions
- Nearly always write functions in a file

BASICS OF AGGROUPS

- AG is for "auflösbare Gruppen" – soluble groups
- In GAP4 we use Pc for polycyclic instead (and can handle infinite groups)
- Elements are represented by `AgWords` – multiplication uses collection
- To make an AG Group:
  - Convert an existing finite soluble group (eg from permutations), using `AgGroup`
  - Create a basic group from the library with `AgWords` as first argument
  - Convert a FP group which actually is a *consistent* pc-presentation, using `AgGroupFpGroup`
  - Compute a p- or soluble quotient of an FP group
  - Form subgroups of other Ag Groups
  - Extract groups from the libraries of groups of small order

Making AgGroups 1

- From existing soluble groups:

```
gap> s := SymmetricGroup(8);
Group( (1,8), (2,8), (3,8), (4,8), (5,8), (6,8), (7,8) )
gap> s1 := Stabilizer(s,[[1,2],[3,4],[5,6],[7,8]],OnSetSet
gap> Size(s1);
384
gap> a := AgGroup(s1);
Group( g1, g2, g3, g4, g5, g6, g7, g8 )
gap> List(a.generators,RelativeOrderAgWord);
[ 2, 3, 2, 2, 2, 2, 2, 2 ]
gap> List(a.generators,x->Image(a.bijection,x));
[ (1,3)(2,4), (1,6,3,2,5,4), (1,5)(2,6)(3,8)(4,7), (1,4)(2
  (1,2), (3,4), (5,6), (7,8) ]
```

Making AgGroups 2

- From the library

```
gap> SymmetricGroup(AgWords,4);
Group( g1, g2, g3, g4 )
```

- From a FP group

```
gap> f := FreeGroup(4);
Group( f.1, f.2, f.3, f.4 )
gap> $= f.4;;
Group( a, b, c, d )
gap> f/[a^2,b^3,c^2,d^2, Comm(a,b)/b^-1, Comm(a,c)/d,Comm(
> Comm(b,d)/(c*d), Comm(c,d)];
Group( a, b, c, d )
gap> AgGroupFpGroup(last);
Group( a, b, c, d )
gap> Size(last);
24
```

- Note that the presentation must be a consistent pc-presentation

Making Ag Groups 3

- There is a p-quotient procedure in GAP `pQuotient`
- Also a solvable quotient `SolvableQuotient`
- There are also links to the ANU p and soluble quotient programs
- See the manual pages for details
- Some small work is needed to turn the output into an AgGroup

Making Ag Groups 4

- GAP contains a library of all groups of order up to 1000, except orders 512 and 768
- Most are solvable, and stored as AgGroups

```
gap> g := SmallGroup(384,1001);
384_1001
gap> List(CompositionSeries(g),Size);
[ 384, 192, 96, 48, 24, 12, 6, 3, 1 ]
gap> Size(Centre(g));
4
gap> IdGroup(g);
[ 384, 1001 ]
```

- `IdGroup` identifies the isomorphism class of a group

Special Ag Groups

- Ideas due to Charles Leedham-Green and Bettina Eick
- A specially nice choice of Ag generators for a group
- Once you have this many operations become much quicker
- Compute it using `SpecialAgGroup`