GAP

Steve Linton
Division of Computer Science, St. Andrews

July 7, 1997

INTRODUCTION

Overview

- What is GAP? – history, purpose, availability, etc.
- Capabilities, practical limits
- Structure, organization
- Comparison with "typical" CA systems
- Future Directions

What is GAP?

- Groups, Algorithms, Programming
- Integrated system for computational group theory, spreading to combinatorics, algebra and discrete mathematics
- Free software, covered by FSF-like Copyleft
- Runs on UNIX, MS-DOS, MacOS, TOS, easy to port
- Developed since 1984 at Lehrstuhl D für Mathematik, RWTH-Aachen
- J. Neubüser, M. Schönert plus cast of thousands
- Increasing external involvement
- Centre of development transferring to St. Andrews in 1997

Structure of GAP

Inspired by that of Maple

kernel written in C (70K lines), implements

- the GAP language;
- system functions;
- "arithmetic" for built-in types;
- some accelerators
- user interface

library written in GAP (140K lines),

- implements most functions
- centrally supported
- "object-oriented" structure – property of library, not the language

databases character tables, groups of order dividing 256 or 729, primitive groups of small degree, crystallographic groups, . . .

share packages user-supported additions, some pure GAP, some links to external programs, one graphical user interface (XGAP)

CAPABILITIES OF GAP

Permutation Groups

- Orbits, primitivity, “can we map this to that?”
- Group order, centralizer, normalizer, composition series, conjugacy classes, Sylow subgroups, . . .
- Conversion to finitely presented groups, p-groups, matrix groups
- database of small degree primitive groups
- some functions good up to degree 1 million or more

Finitely-presented Groups

- Coset enumeration, Reidermeister-Schreier, Low index, etc.
- Abelian, nilpotent and soluble quotients
- Tietze transformations simplifying presentations

Finite Polycyclic (Soluble) Groups

- Usually constructed from FP or permutation groups – stored as power-commutator presentation, manipulated using commutator collection.
- Group order, centralizer, normalizer, composition series, conjugacy classes, Sylow subgroups, Hall subgroups, maximal subgroups, . . .
- database of groups of order dividing 256 or 729

Character Theory

- Database of all ATLAS 1 and 2 character tables
- Many functions to compute with them – restriction, fusion, inner products, structure constants, embeddings, monomiality, etc.
- GAP much used in determining ATLAS 2 tables. Powerful methods for ordinary and modular tables.

Share Packages

- ANU p-Quotient – E. O'Brien
- ANU soluble-Quotient – A. Niemeyer
- GRAPE – functions for graphs, studied via groups of automorphisms – Len Soicher
- GUAVA – functions for coding theory – Delft university of Technology
- Meat-axe – matrix groups, representation theory –stand-alone by R. Parker, link-up by T. Breuer
- Sisyphos – group rings of p-groups
- Smash – new methods for decomposition of matrix representations – S. Rees, D. Holt, C.R. Leedham-Green, E. O'Brien
- Vector enumeration – linear Todd-Coxeter (SL)
- Weyl – Weyl groups and Hecke algebras – G. Malle, M. Geck
- DCE – double coset enumeration – SL

- KBMAG – Knuth-Bendix and Automatic Groups – D. Holt
- XGAP – Windowing user interface – F. Celler

CS ASPECTS OF GAP

The GAP language

- Interpreted language
- "Standard" modern imperative language,
- functions are first class objects
- Untyped
- Built-in arithmetic for:
  - integers
  - rationals
  - cyclotomic integers and rationals
  - finite field elements
  - permutations
  - words in abstract generators
- Built-in data structure:
  - lists (really flexible arrays)
  - records
  - vectors
  - sets (stored as lists)

Domains and Categories

The GAP language provides (almost) no type system and no explicit support for object-oriented methods. There is no overloading of function names, for example.

Nevertheless the GAP library is almost entirely object-oriented.

One computes with:

Elements – such as integers, permutations, words or user-defined things

Domains – structured sets such as groups, cosets, rings, fields etc.

Categories – sets of domains such as "all groups" – organized in a hierarchy of sub and super categories eg `SymmetricGroups` in `PermutationGroups` in `Groups`.

Most "high-level" computation goes on with domains, reduced by the library to computations with elements.

Most algorithms are written for a category

Dispatchers and Operations Records

Each domain (and many elements) are represented as records, containing a component `operations` which is itself a record, containing methods applicable to that object.

Many globally defined functions then refer to the methods contained in their arguments.

For example, in:

```
gap> a5 := AlternatingGroup(5);
Group( (1,2,5), (2,3,5), (3,4,5) )
gap> Size(a5);
60
```

- `a5` is a record.
- Global function `Size`:
  - checks if the field `a5.size` is present,
  - if not calls `a5.operations.Size(a5)` to compute it (and sets `a5.size`).
- Such functions are called dispatchers.

Operations Records and Categories

In principle every domain has its own operations record, but in practice, these are shared between domains in the same (smallest) category.

There is a global record `SymmetricGroupOps` and all symmetric groups contain a pointer to it.

`SymmetricGroupOps` is created in the library by taking a copy of `PermGroupOps`, so there is effective *inheritance* of all permutation group methods by symmetric groups.

A member of `SymmetricGroupOps` can also handle some cases specially for symmetric groups, but otherwise call the equivalent member of `PermGroupOps`. A process called *delegation*.

GAP vs. a Truly O-O System

- GAP is more flexible (with attendant risks of confusion)
- Kernel remains small and simple
- Method selection cannot be done until run time
- Certain operations `()`, `:=`, etc. not programmable
- No data hiding in GAP (except by convention)
- No automatic checking of O-O conventions
- Categories and classes are used rather differently
- Some run-time overhead

GAP AS A COMPUTER ALGEBRA SYSTEM

GAP as a Computer Algebra System

In comparison with "typical" CAS's GAP:

- emphasises different areas of mathematics – algebra, combinatorics, number theory vs analysis (calculus), algebraic geometry
- lacks the very general "expression" type that most CAS's have, but has (for example) polynomials
- has more of a type system that Maple or Mathematica, but less of one that Axiom
- (in most areas) emphasises performance for the serious research user over simplicity for casual use (when these cannot be reconciled)
- has remarkably few bugs

FUTURE DIRECTIONS

GAP 4 is under development now in Aachen. The system is being reengineered from the ground up:

- New memory manager
- New long integer arithmetic
- Faster function calling
- More modular kernel
- Possible compiler (GAP$->$ C, converting library functions into kernel functions)
- Dispatchers moved into the kernel
- New library loading scheme – quicker and more memory efficient
- More even-handed treatment of different algebraic structures – less of a special role for groups
- iterators and enumerators

Compiler

GAP programs that don't use large-scale kernel functions (like multiplying large permutations) tend to run about 10 times slower than their C equivalents. This is roughly:

- a factor of 3 due to the interpreter
- a factor of 3 due to the memory management (no real pointers)

The compiler removes the first delay by converting GAP into C which can then be linked with the running kernel. Some additional optimisation may be possible.

The new memory manager should reduce the second delay somewhat.

Method Selection

- The system of dispatchers and inheritance will be replaced by a system based on *property lists*.
- Each object has a list of properties such as `IsGroup`, `IsSoluble`, `IsFinite`, `IsNilpotent`, which may be true, false or unknown. Setting some properties automatically sets others.
- Each operation has a number of methods installed, each of which requires that its arguments have certain properties.
- This allows method selection based on more than one argument, and other combinations not available in a standard O-O model.
- `[]`, `.`, `:=`, etc. now programmable