GAP

Christoph Koutschan

Research Institute for Symbolic Computation
Johannes Kepler Universität Linz, Austria

Computer Algebra Systems
29.11.2010

GAP

- **GAP** = **G**roups, **A**lgorithms and **P**rogramming
- freely available open source software
  (subject to copyleft conditions)
- sources, packages, data library (400MB in total)
- user contributed packages:
  - important feature of the system
  - GAP offers authors to submit packages
  - peer review
  - ca. 35 of 58 packages were included this way
- `http://www.gap-system.org`

History

- 1986–1997: developed at Lehrstuhl D für Mathematik (LDFM), RWTH Aachen
- 1997: retirement of founder Joachim Neubüser
- 1997–2005: maintenance and development coordinated by School of Mathematical and Computational Sciences at the University of St. Andrews, Scotland.
- since 2005: Four GAP Centres:
  - University of St. Andrews
  - RWTH Aachen
  - Technische Universität Braunschweig
  - Colorado State University at Fort Collins

The GAP group at ISSAC'2008 in Linz

Photo: Christoph Koutschan

First Steps in GAP

```
gap> a := 5*6^7;
1399680
gap> a := 9^9;;
gap> a;
387420489
gap> a = 9*9;
false
gap> NamesGVars();
...
```

Lists

```
gap> list := [2,4,7];
[ 2, 4, 7 ]
gap> list[2];
4
gap> list[2] := 5;; list;
[ 2, 5, 7 ]
gap> Add(list, 1);
gap> list;
[ 2, 5, 7, 1 ]
gap> Product(list);
70
gap> 2*list;
[ 4, 10, 14, 2 ]
gap> list^2;
79
gap> Append(list, list); list;
[ 2, 5, 7, 1, 2, 5, 7, 1 ]
```

Sorting

```
gap> Sort([5,3,4]);
gap> list := [5,3,4];
[ 5, 3, 4 ]
gap> Sort(list);
gap> list;
[ 3, 4, 5 ]
gap> list := [5,3,4];
[ 5, 3, 4 ]
gap> Sortex(list);
(1,3,2)
gap>
```

Permutations

```
gap> (1,2,3);
(1,2,3)
gap> (2,3,1);
(1,2,3)
gap> (1,2,3)^-1;
(1,3,2)
gap> (1,2,3)*(1,3,2);
()
gap> 3^(1,3,2);
2
```

Objects versus Elements

- every object is given a certain place in memory by the GAP storage manager
- objects at different places in memory are never equal

```
gap> a:= (1,2);; IsIdenticalObj( a, a );
true
gap> b:= (1,2);; IsIdenticalObj( a, b );
false
gap> b:= a;; IsIdenticalObj( a, b );
true
```

- GAP uses the equality operator = to denote mathematical equality
- the operator = defines an equivalence relation on the set of all GAP objects
- the corresponding equivalence classes are called elements
- the same element may be represented by various GAP objects

Groups

```
gap> S6 := SymmetricGroup(6);
Sym( [ 1 .. 6 ] )
gap> Size(S6);
720
gap> s6 := Group( (1,2), (1,2,3,4,5,6) );
Group([ (1,2), (1,2,3,4,5,6) ])
gap> IsIdenticalObj(S6,s6);
false
gap> S6 = s6;
true
gap> IsAbelian(s6);
false
gap> IsAssociative(s6);
true
```

Subgroups

```
gap> sub6 := Subgroup(S6, [(1,2,3,4,5,6)]);
Group([ (1,2,3,4,5,6) ])
gap> Index(S6,sub6);
120
gap> Index(S6, Group((1,2,3,4,5,6)));
120
gap> Index(S6, SymmetricGroup(4));
30
gap> AsList(SymmetricGroup(4));
[ (), (3,4), (2,3), (2,3,4), (2,4,3), (2,4), (1,2),
  (1,2)(3,4), (1,2,3), (1,2,3,4), (1,2,4,3), (1,2,4),
  (1,3,2), (1,3,4,2), (1,3), (1,3,4), (1,3)(2,4),
  (1,3,2,4), (1,4,3,2), (1,4,2), (1,4,3), (1,4),
  (1,4,2,3), (1,4)(2,3) ]
```

Commutator Subgroups

```
gap> a6 := DerivedSubgroup(s6);
Group([ (1,2,3), (2,3,4), (2,4)(3,5), (2,6,4) ])
gap> Size(a6);
360
gap> (1,2) in a6;
false
gap> DerivedSubgroup(a6) = a6;
true
gap> IsPerfect(a6);
true
gap> A6 := AlternatingGroup(6);
Alt( [ 1 .. 6 ] )
gap> A6 = a6;
true
```

Sign of a Permutation

```
gap> SignPerm((1,2,3));
1
gap> SignPerm((1,2));
-1
gap> List(AsList(a6),SignPerm);
[ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

Free Groups

```
gap> fg := FreeGroup("a","b");
<free group on the generators [ a, b ]>
gap> Size(fg);
infinity
gap> x := fg.1; y := fg.2;
a
b
gap> y*y*y;
b^3
```

Free Groups

```
gap> rels := [x^2,y^3,x*y*x*y];
[ a^2, b^3, a*b*a*b ]
gap> fg1 := fg / rels;
<fp group on the generators [ a, b ]>
gap> Size(fg1);
6
gap> AsList(fg1);
[ <identity ...>, a, b, a*b, a*b*a, b*a ]
gap> y in fg1;
false
gap> x := fg1.1;; y := fg1.2;;
gap> x*x*x = x;
true
gap> MultiplicationTable(fg1);
[ [ 1, 2, 3, 4, 5, 6 ], [ 2, 1, 4, 3, 6, 5 ],
  [ 3, 6, 5, 2, 1, 4 ], [ 4, 5, 6, 1, 2, 3 ],
  [ 5, 4, 1, 6, 3, 2 ], [ 6, 3, 2, 5, 4, 1 ] ]
```

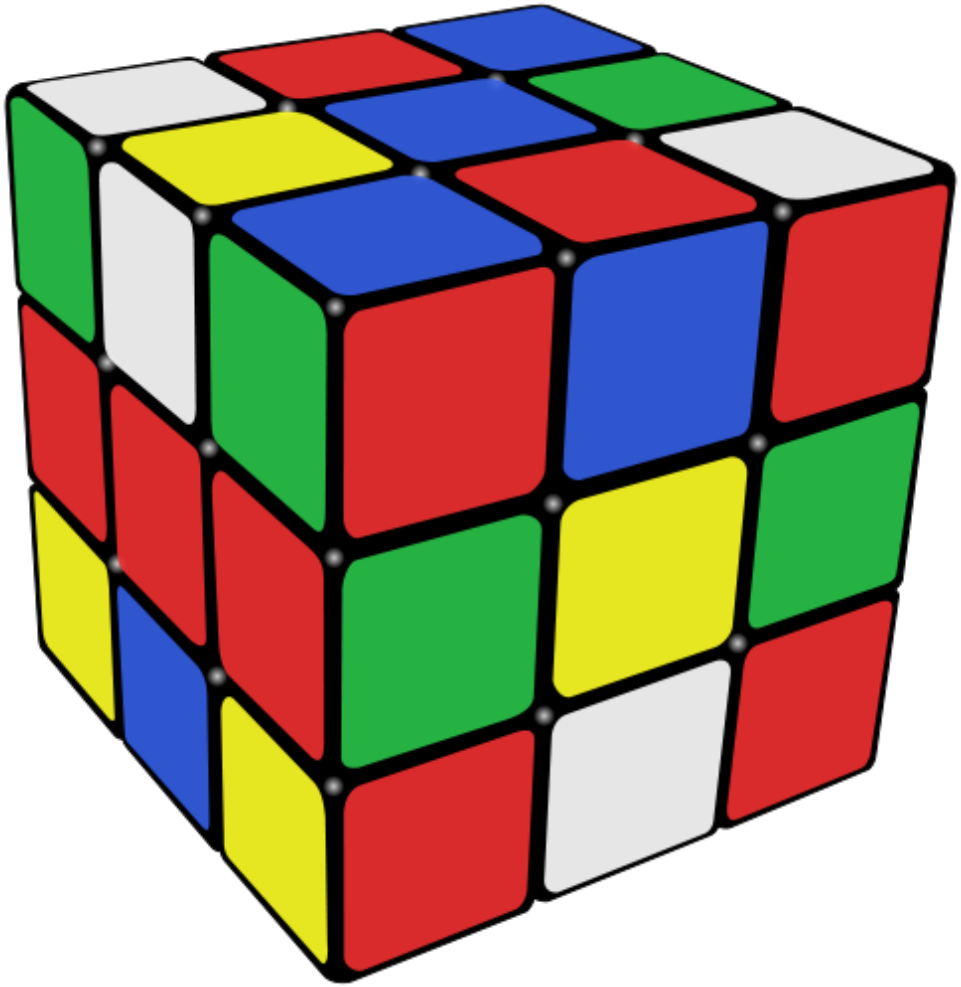
Vector Spaces

```
gap> V:= VectorSpace(Rationals, [[1, 1, 1], [1, 0, 2]]);
<vector space over Rationals, with 2 generators>
gap> [2, 1, 3] in V;
true
gap> Dimension(V);
2
gap> IsVectorSpace(V);
true
gap> IsVectorSpace(Rationals);
true
gap> Basis(V);
SemiEchelonBasis( <vector space over Rationals,
  with 2 generators>, ... )
gap> BasisVectors(Basis(V));
[ [ 1, 1, 1 ], [ 0, 1, -1 ] ]
```

Vector Spaces

```
gap> V:= GF(5)^3;
( GF(5)^3 )
gap> [2, 1, 3] in V;
false
gap> AsList(GF(5));
[ 0*Z(5), Z(5)^0, Z(5), Z(5)^2, Z(5)^3 ]
gap> [Z(5),Z(5)^0,Z(5)^2] in V;
true
```

Case Study: Rubik's Cube



Rubik's Cube

Question: How many moves are necessary to solve an arbitrarily scrambled cube?

- "human algorithms": ~ 100 moves
- 1981: Thistlethwaite's algorithm: 52 moves
- 1995: Michael Reid: 29 (face) moves or 42 (quarter) moves
- 2005: Silviu Radu: 40 quarter moves
- 2006: Silviu Radu: 27 face turns or 35 quarter turns
- 2010: Morley Davidson: all cube positions can be solved with at most 20 face turns

$\longrightarrow$ Some of these results were obtained with GAP.

Moves for Rubik's Cube

```
vorne  := (1,3,5,7)(2,4,6,8)(23,9,37,47)(22,16,38,48)
          (21,15,39,41);
hinten:= (25,27,29,31)(26,28,30,32)(17,11,35,45)
          (18,12,34,44)(19,13,33,43);
oben   := (17,19,21,23)(18,20,22,24)(1,43,27,9)
          (2,42,26,10)(3,41,25,11);
unten  := (33,35,37,39)(34,36,38,40)(5,47,31,13)
          (6,46,30,14)(7,45,29,15);
rechts:= (9,11,13,15)(10,12,14,16)(3,19,29,37)
          (4,20,28,36)(5,21,27,35);
links  := (41,43,45,47)(42,44,46,48)(1,17,31,39)
          (8,24,32,40)(7,23,25,33);
```

```
gap> rubik := Group(vorne,hinten,oben,unten,rechts,links);
<permutation group with 6 generators>
gap> Size(rubik);
43252003274489856000
gap> Factorial(8)*Factorial(12)*3^8*2^12;
519024039293878272000
gap> last / Size(rubik);
12
```

$\longrightarrow$ Schreier-Sims algorithm makes such computations possible via stabilizer chains.

```
gap> (3,4) in rubik
false
gap> (3,9,21) in rubik
false
gap> (2,22) in rubik
false
gap> gens := List(GeneratorsOfGroup(rubik));;
gap> Append(gens, [(2,22),(3,9,21),(21,19)(3,11)(9,27)]);
gap> Size(Group(gens));
519024039293878272000
gap> (20,22)(2,10) in rubik;
false
gap> (20,22)(2,10) in Group(gens);
true
```

```
gap> DerivedSeries(rubik);
[ <permutation group of size 43252003274489856000 with 6
generators>,
  <permutation group of size 2162600163724492800 with 5
generators> ]
gap> Centre(rubik);
Group([
(2,22)(4,16)(6,38)(8,48)(10,20)(12,28)(14,36)(18,26)
    (24,42)(30,34)(32,44)(40,46) ])
gap> Random(rubik);
(1,9,33,13,5,17,39)(2,26,4,30,40,22,18,16,34,46)
    (3,31,29,15,43,47,41)(6,48,10,38,8,20)
    (7,23,21,45,35,37,25)(12,14,28,36)(24,32,42,44)
```

```
gap> cent := Centralizer(rubik, Group((1,2),
       (1,2,3,4,5,6,7,8,25,26,27,28,29,30,31,32)));
<permutation group with 3 generators>
gap> Size(cent);
96
gap> GeneratorsOfGroup(cent);
[ (24,42)(40,46), (14,24,40)(36,42,46),
  (10,14)(20,36)(24,40)(42,46) ]
```

```
gap> norm := Normalizer(rubik, Group((1,2),
       (1,2,3,4,5,6,7,8,25,26,27,28,29,30,31,32)));
<permutation group with 17 generators>
gap> stab := Stabilizer(rubik, Group((1,2),
       (1,2,3,4,5,6,7,8,25,26,27,28,29,30,31,32)));
<permutation group with 17 generators>
gap> List([norm,stab],Size);
[ 156067430400, 156067430400 ]
gap> norm = stab;
true
```

```
gap> IsCyclic(rubik);
false
gap> SmallGeneratingSet(rubik);
[ (1,39,29,27,45,23,7,13,19,31,41,47,35,11,33)
  (2,36,32,38,4,28,24,46)(3,43)(5,15,37)
  (6,16,12,42,40,22,14,44)(8,34,20)(9,17)(10,48,30)
  (21,25),
  (1,39,21)(2,44,38,16,48,20,26,22,32,6,4,8,10,18)
  (3,23,7)(5,43,15,25,37,17)(9,41,47)(11,29,19,35,27,13)
  (12,14,28,36)(24,34,42,30)(31,45,33)(40,46) ]
gap> Length(last);
2
```