Computer Science Journal of Moldova, vol.**1**, no.3(3), 1993

An Invitation to GAP

M.Schönert

1 Introduction

GAP is a system for computational discrete algebra, which we have developed with particular emphasis on computational group theory, but which has already proved useful also in other areas. The name GAP is an acronym for **Groups, Algorithms, and Programming**. This document announces the availability of GAP version 3 release 2, GAP 3.2 for short. It is an **advertisement** for GAP, but not a **commercial**, since we give GAP away for free.

This document begins with the section "Analyzing Rubik's Cube with GAP", which contains an extensive example. This example is followed by a general discussion of GAP's capabilities in the section "An Overview of GAP". The section "Copyright" states the terms under which you can copy GAP. The next section "How to get GAP" describes how you can get GAP. Then we tell you about our plans for the future in the section "The Future of GAP". The final section "The GAP Forum" introduces the GAP forum, where interested users can discuss GAP related topics by e-mail messages.

GAP has been developed at the Lehrstuhl D für Mathematik since 1986. Many people, mostly students of our institute have contributed to it:

Alice Niemeyer, Werner Nickel, Martin Schönert, Johannes Meier, Alex Wegner, Thomas Bischops, Frank Celler, Jürgen Mnich, Udo Polis, Thomas Breuer, Götz Pfeiffer, Hans U. Besche, Volkmar Felsch, Heiko Theissen, Alexander Hulpke, Ansgar Kaup, Akos Seress.

2 Analyzing Rubiks Cube with GAP

> Ideal Toy Company stated on the package of
> the original Rubik cube that there were more than
> three billion possible states the cube could attain.
> It's analogous to Mac Donald's proudly announcing
> that they've sold more than 120 hamburgers.
> (J. A. Paulos, Innumeracy)

To show you what GAP can do a short example is probably best. If you are not interested in this example skip to the section "An Overview of GAP".

For the example we consider the group of transformations of Rubik's magic cube. If we number the faces of this cube as follows

```
              +------------+
              !  1   2   3 !
              !  4   U   5 !
              !  6   7   8 !
+-------------+------------+------------+------------+
!  9  10  11 ! 17  18  19 ! 25  26  27 ! 33  34  35 !
! 12   L  13 ! 20   F  21 ! 28   R  29 ! 36   B  37 !
! 14  15  16 ! 22  23  24 ! 30  31  32 ! 38  39  40 !
+-------------+------------+------------+------------+
              ! 41  42  43 !
              ! 44   D  45 !
              ! 46  47  48 !
              +------------+
```

then the group is generated by the following generators, corresponding to the six faces of the cube (the two semicolons tell GAP not to print the result, which is identical to the input here).

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

First we want to know the size of this group.

```
gap> Size( cube );
43252003274489856000
```

Since this is a little bit unhandy, let us factorize this number.

```
gap> Collected( Factors( last ) );
[ [ 2, 27 ], [ 3, 14 ], [ 5, 3 ], [ 7, 2 ], [ 11, 1 ] ]
```

(The result tells us that the size is $2^{27}3^{14}5^{3}7^{2}11$.)

Next let us investigate the operation of the group on the 48 points.

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

The first orbit contains the points at the corners, the second those at the edges; clearly the group cannot move a point at a corner onto a point at an edge.

So to investigate the cube group we first investigate the operation on the corner points. Note that the constructed group that describes

this operation will operate on the set [1..24], not on the original set [1,3,17,14,8,38,9,41,19,48,22,6,30,33,43,11,46,40,24,27,25,35,16,32].

```
gap> cube1 := Operation( cube, orbits[1] );
Group( ( 1, 2, 5,12)( 3, 7,14,21)( 9,16,22,20),
       ( 1, 3, 8,18)( 4, 7,16,23)(11,17,22,12),
       ( 3, 9,19,11)( 5,13, 8,16)(12,21,15,23),
       ( 2, 6,15, 9)( 5,14,10,19)(13,21,20,24),
       ( 1, 4,10,20)( 2, 7,17,24)( 6,14,22,18),
       ( 4,11,13, 6)( 8,15,10,17)(18,23,19,24) )
gap> Size( cube1 );
88179840
```

Now this group obviously operates transitively, but let us test whether it is also primitive.

```
gap> corners := Blocks( cube1, [1..24] );
[ [ 1, 7, 22 ], [ 2, 14, 20 ], [ 3, 12, 16 ],
  [ 4, 17, 18 ], [ 5, 9, 21 ], [ 6, 10, 24 ],
  [ 8, 11, 23 ], [ 13, 15, 19 ] ]
```

Those eight blocks correspond to the eight corners of the cube; on the one hand the group permutes those and on the other hand it permutes the three points at each corner cyclically.

So the obvious thing to do is to investigate the operation of the group on the eight corners.

```
gap> cube1b := Operation( cube1, corners, OnSets );
Group( (1,2,5,3), (1,3,7,4), (3,5,8,7),
       (2,6,8,5), (1,4,6,2), (4,7,8,6) )
gap> Size( cube1b );
40320
```

Now a permutation group of degree 8 that has order 40320 must be the full symmetric group $S(8)$ on eight points.

The next thing then is to investigate the kernel of this operation on blocks, i.e., the subgroup of `cube1` of those elements that fix the blocks setwise.

```
gap> blockhom1 := OperationHomomorphism( cube1, cube1b );;
gap> Factors( Size( Kernel( blockhom1 ) ) );
[ 3, 3, 3, 3, 3, 3, 3 ]
gap> IsElementaryAbelian( Kernel( blockhom1 ) );
true
```

We can show that the product of this elementary abelian group 3^7 with the $S(8)$ is semidirect by finding a complement, i.e., a subgroup that has trivial intersection with the kernel and that generates `cube1` together with the kernel.

```
gap> cmpl1:=Stabilizer(cube1,[1,2,3,4,5,6,8,13],OnSets);;
gap> Size( cmpl1 );
40320
gap> Size( Intersection( cmpl1, Kernel( blockhom1 ) ) );
1
gap> Closure( cmpl1, Kernel( blockhom1 ) ) = cube1;
true
```

There is even a more elegant way to show that `cmpl1` is a complement.

```
gap> IsIsomorphism( OperationHomomorphism(cmpl1,cube1b) );
true
```

Of course, theoretically it is clear that `cmpl1` must indeed be a complement.

In fact we know that `cube1` is a subgroup of index 3 in the wreath product of a cyclic 3 with $S(8)$. This missing index 3 tells us that we do not have total freedom in turning the corners. The following tests show that whenever we turn one corner clockwise we must turn another corner counterclockwise.

```
gap> (1,7,22) in cube1;
false
gap> (1,7,22)(2,20,14) in cube1;
true
```

More or less the same things happen when we consider the operation of the cube group on the edges.

```
gap> cube2 := Operation( cube, orbits[2] );;
gap> Size( cube2 );
980995276800
gap> edges := Blocks( cube2, [1..24] );
[ [ 1, 11 ], [ 2, 17 ], [ 3, 19 ], [ 4, 22 ], [ 5, 13 ],
  [ 6, 8 ], [ 7, 24 ], [ 9, 18 ], [ 10, 21 ], [ 12, 15 ],
  [ 14, 20 ], [ 16, 23 ] ]
gap> cube2b := Operation( cube2, edges, OnSets );;
gap> Size( cube2b );
479001600
gap> blockhom2 := OperationHomomorphism( cube2, cube2b );;
gap> Factors( Size( Kernel( blockhom2 ) ) );
[ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
gap> IsElementaryAbelian( Kernel( blockhom2 ) );
true
gap> cmpl2 := Stabilizer( cube2,
>                [1,2,3,4,5,6,7,9,10,12,14,16], OnSets );;
gap> IsIsomorphism( OperationHomomorphism(cmpl2,cube2b) );
true
```

This time we get a semidirect product of a 2^{11} with an $S(12)$, namely a subgroup of index 2 of the wreath product of a cyclic 2 with $S(12)$. Here the missing index 2 tells us again that we do not have total freedom in turning the edges. The following tests show that whenever we flip one edge we must also flip another edge.

```
gap> (1,11) in cube2;
false
gap> (1,11)(2,17) in cube2;
true
```

Since `cube1` and `cube2` are the groups describing the actions on the two orbits of `cube`, it is clear that `cube` is a subdirect product of those groups, i.e., a subgroup of the direct product. Comparing the sizes of `cube1`, `cube2`, and `cube` we see that `cube` must be a subgroup of index 2 in the direct product of those two groups.

```
gap> Size( cube );
43252003274489856000
gap> Size( cube1 ) * Size( cube2 );
86504006548979712000
```

This final missing index 2 tells us that we cannot operate on corners and edges totally independently. The following tests show that whenever we exchange a pair of corners we must also exchange a pair of edges (and vice versa).

```
gap> (17,19)(11,8)(6,25) in cube;
false
gap> (7,28)(18,21) in cube;
false
gap> (17,19)(11,8)(6,25)(7,28)(18,21) in cube;
true
```

Finally let us compute the centre of the cube group, i.e., the subgroup of those operations that can be performed either before or after any other operation with the same result.

```
gap> Centre( cube );
Subgroup( cube, [ ( 2,34)( 4,10)( 5,26)( 7,18)(12,37)
    (13,20)(15,44)(21,28)(23,42)(29,36)(31,45)(39,47) ] )
```

We see that the centre contains one nontrivial element, namely the operation that flips all 12 edges simultaneously.

This concludes our example. Of course, GAP can do much more, and the next section gives an overview of its capabilities, but demonstrating them all would take too much room.

3 An Overview of GAP

Though this be madness,
yet there is method in't.
(W. Shakespeare, Hamlet)

GAP consists of several parts: the kernel, the library of functions, the library of groups and related data, and the documentation.

The **kernel** implements an automatic memory management, a PASCAL like programming language, also called GAP, with datatypes for computations in group theory, and an interactive programming environment to run programs written in the GAP programming language.

The automatic **memory management** allows programmers to concentrate on implementing the algorithm without needing to care about allocation and deallocation of memory. It includes a garbage collection that automatically throws away objects that are no longer accessible.

The GAP programming language supports a number of datatypes for elements of fields. **Integers** can be arbitrarily large, and are implemented in such a way that operations with small integers are reasonably fast. Building on this large-integer arithmetic GAP supports **rationals** and elements from **cyclotomic fields**. Also GAP allows one to work with elements from **finite fields** of size (at present) at most 2^{16}.

The special datatypes of group elements are **permutations**, **matrices** over the rationals, cyclotomic fields, and finite fields, **words in abstract generators**, and **words in solvable groups**.

GAP also contains a very flexible **list** datatype. A list is simply a collection of objects that allows you to access the components using an integer position. Lists grow automatically when you add new elements to them. Lists are used to represent sets, vectors, and matrices. A **set** is represented by a sorted list without duplicates. A list whose elements all lie in a common field is a **vector**. A list of vectors of the same length over a common field is a **matrix**. Since sets, vectors, and matrices are lists, all list operations and functions are applicable. You can, for

example, find a certain element in a vector with the general function `Position`. There are also **ranges**, i.e., lists of consecutive integers, and **boolean lists**, i.e., lists containing only `true` and `false`. Vectors, ranges, and boolean lists have special internal representations to ensure efficient operations and memory usage. For example, a boolean list requires only one bit per element.

Records in GAP are similar to lists, except that accessing the components of a record is done using a name instead of an index. Records are used to collect objects of different types, while lists usually only contain elements of one type. Records are for example used to represent groups and other domains; there is **no** group datatype in the GAP language . Because of this all information that GAP knows about a group is also accessible to you by simply investigating the record.

The control structures of GAP are PASCAL-like. GAP has **if** statements, **while**, **repeat**, and **for** loops. The for loop is a little bit uncommon in that it always loops over the elements of a list. The usual semantics can be obtained by looping over the elements of a range. Using those building blocks you can write **functions**. Functions can be recursive, and are first class objects in the sense that you can collect functions in lists, pass them as arguments to other functions and also return them.

It is important to note that GAP has dynamic typing instead of static typing. That means that the datatype is a property of the object, not of the variable. This allows you to write general functions. For example the generic function that computes an orbit can be used to compute the orbit of an integer under a permutation group, the orbit of a vector under a matrix group, the conjugacy class of a group element, and many more.

The kernel also implements an **interactive environment** that allows you to use GAP. This environment supports debugging; in case of an error a break loop is entered in which you can investigate the problem, and maybe correct it and continue. You also have online access to the manual, though sections that contain larger formulas do not look nice on the screen.

The **library of functions**, simply called library in the following, contains implementations of various group theoretical algorithms written in the GAP language. Because all the group theoretical functions are in this library it is easy for you to look at them to find out how they work, and change them if they do almost, but not quite, what you want.

The whole library is centered around the concept of domains and categories. A **domain** is a structured set, e.g., a group is a domain as is the ring of Gaussian integers. Each domain in GAP belongs to one or more **categories**, which are simply sets of domains, e.g., the set of all groups forms a category. The categories in which a domain lies determine the functions that are applicable to this domain and its elements.

To each domain belongs a set of functions, in a so called operations record, that are called by dispatchers like `Size`. For example, for a permutation group G, G`.operations.Size` is a function implementing the Schreier Sims algorithm. Thus if you have any domain D, simply calling `Size(` D `)` will return the size of the domain D, computed by an appropriate function. Domains **inherit** such functions from their category, unless they redefine them. For example, for a permutation group G, the derived subgroup will be computed by the generic group function, which computes the normal closure of the subgroup generated by the commutators of the generators.

Of course the most important category is the category of **groups**. There are about 100 functions applicable to groups. These include general functions such as `Centralizer` and `SylowSubgroup`, functions that compute series of subgroups such as `LowerCentralSeries`, a function that computes the whole lattice of subgroups, functions that test predicates such as `IsSimple`, functions that are related to the operations of groups such as `Stabilizer`, and many more. Most of these functions are applicable to all groups, e.g., permutation groups, finite polycyclic groups, factor groups, direct products of arbitrary groups, and even new types of groups that you create by simply specifying how the elements are multiplied and inverted (actually it is not quite so simple, but you can do it).

Where the general functions that are applicable to all groups are not

efficient enough, we have tried to overlay them by more efficient functions for special types of groups. The prime example is the category of **permutation groups**, which overlays `Size`, `Elements`, `Centralizer`, `Normalizer`, `SylowSubgroup`, and a few more functions by functions that employ stabilizer chains and backtracking algorithms. Also many of the functions that deal with operations of groups are overlayed for permutation groups for the operation of a permutation group on integers or lists of integers.

Special functions for **finitely presented groups** include functions to find the index of a subgroup via a Todd-Coxeter coset enumeration, to compute the abelian invariants of the commutator factor group, to intersect two subgroups, to find the normalizer of a subgroup, to find all subgroups of small index, and to compute and simplify presentations for subgroups. Of course it is possible to go to a permutation group operating on the cosets of a subgroup and then to work with this permutation group.

For **finite polycyclic groups** a special kind of presentation corresponding to a composition series is used. Such a presentation implies a canonical form for the elements and thus allows efficient operations with the elements of such a group. This presentation is used to make functions such as `Centralizer`, `Normalizer`, `Intersection`, and `ConjugacyClasses` very efficient. GAP's capabilities for finite polycyclic groups exceed those of the computer system SOGOS (which was developed at Lehrstuhl D für Mathematik for the last decade).

There is also support for **mappings** and **homomorphisms**. Since they play such a ubiquitous role in mathematics, it is only natural that they should also play an important role in a system like GAP. Mappings and homomorphisms are objects in their own right in GAP. You can apply a mapping to an element of its source, multiply mappings (provided that the range of the first is a subset of the source of the second), invert mappings (even if what you get is a multi-valued mapping), and perform a few more operations. Important examples are the `NaturalHomomorphism` onto a factor group, `OperationsHomomorphism` mapping a group that operates on a set of n elements into the symmet-

ric group on $[1..n]$, `Embeddings` into products of groups, `Projections` from products of groups onto the components, and the very general `GroupHomomorphismByImages` for which you only specify the images of a set of generators.

The library contains a package for handling **character tables** of finite groups. This includes almost all possibilities of the computer system CAS (which was developed at Lehrstuhl D für Mathematik in the last decade), and many new functions. You can compute character tables of groups, or construct character tables using other tables, or do some calculations within known character tables. You can, for example, compute a list of candidates for permutation characters. Of course there are many character tables (at the moment more than 650 ordinary tables) in the data library, including all those in the ATLAS of finite groups.

For large integers we now also have a package for **elementary number theory**. There are functions in this package to test primality, factor integers of reasonable size, compute the size phi(n) of the prime residue group modulo an integer n, compute roots modulo an integer n, etc. Also based on this there is a package to do calculations in the ring of Gaussian integers.

The library also includes a package for **combinatorics**. This contains functions to find all selections of various flavours of the elements of a set, e.g., `Combinations` and `Tuples`, or the number of such selections, e.g., `Binomial`. Other functions are related to partitions of sets or integers, e.g., `PartitionsSet` and `RestrictedPartitions`, or the number of such, e.g., `NrPartitions` and `Bell`. It also contains some miscellaneous functions such as `Fibonacci` and `Bernoulli`.

The **data library** at present contains the primitive permutation groups of degree up to 50 from C. Sims, the 2-groups of size dividing 256 from E. O'Brien and M. F. Newman, the 3-groups of size dividing 729 from E. O'Brien and C. Rhodes, the solvable groups of size up to 100 from M. Hall, J. K. Senior, R. Laue, and J. Neubüser, a library of character tables including all of the ATLAS, and a library of tables of marks for

various groups. We plan to extend the data library with more data in the future.

Together with GAP 3.2 we now distribute several **share library packages**. Such packages have been contributed by other authors, but the copyright remains with the author. Currently (May 1993) there are three packages in the share library. The **ANU PQ** package, written by E. O'Brien, consists of a C program implementing a p-quotient and a p-group generation algorithm and functions to interface this program with GAP (or Cayley). The **NQ** package, written by W. Nickel, consists of a C program implementing an algorithm to compute nilpotent quotients of finitely presented groups and a function to call this program from GAP. The **Weyl** package, written by M. Geck, contains functions to compute with finite Weyl groups, associated (Iwahori-) Hecke algebras, and their representations.

4 Copyright

Ceterum censeo:
Nobody has ever paid a licence fee
for using a proof
that shows Sylow's subgroups to exist.
Nobody should ever pay a licence fee
for using a program
that computes Sylow's subgroups.
(J. Neubüser)

If you copy GAP for somebody else, you may ask this person for refund of your expenses. This should cover cost of media, copying and shipping. You are not allowed to ask for more than this. In any case you must give a copy of this copyright notice along with the program.

If you publish a mathematical result that was partly obtained using GAP, please cite GAP, just as you would cite another paper that you used. Also we would appreciate it if you could inform us about such a paper.

You are permitted to modify and redistribute GAP, but you are not allowed to restrict further redistribution. That is to say proprietary modifications will not be allowed. We want all versions of GAP to remain free.

If you modify any part of GAP and redistribute it, you must supply a `README` document. This should specify what modifications you made in which files. We do not want to take credit or be blamed for your modifications.

GAP is distributed by us without any warranty, to the extent permitted by applicable state law. We distribute GAP **as is** without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

The entire risk as to the quality and performance of the program is with you. Should GAP prove defective, you assume the cost of all necessary servicing, repair or correction.

In no case unless required by applicable law will we, and/or any other party who may modify and redistribute GAP as permitted above, be liable to you for damages, including lost profits, lost monies or other special, incidental or consequential damages arising out of the use or inability to use GAP.

The system dependent part of GAP for the 386 (`sysdos.c`) was written by Steve Linton (111 Ross St., Cambridge, CB1 3BS, UK, +44 223 411661, `sl25@cus.cam.ac.uk`). He assignes the copyright to the

Lehrstuhl D fuer Mathematik. Many thanks to Steve Linton for his work.

GAP for the 386 was compiled with DJ Delorie's port of the Free Software Foundation's GNU C compiler version 2.1. The compiler can be obtained by anonymous `ftp` from `grape.ecs.clarkson.edu` where it is in the directory `pub/msdos/djgpp`. Many thanks to the Free Software Foundation and DJ Delorie for this amazing piece of work.

The modifications to the compiler to make it operating under MS-DOS, the functions from the standard library `libpc.a`, the modifications of the functions from the standard library `libc.a` to make them operate under MS-DOS, and the DOS extender `go32` (which is prepended to `gapexe.386`) are

The functions in `libc.a` that GAP for the 386 uses are

5 How to get GAP

GAP is distributed **free of charge**. You can obtain it via `ftp` or electronic mail and give it away to your colleagues.

If you get GAP, we would appreciate it if you could notify us, e.g., by sending a short e-mail message to `gap@samson.math.rwth-aachen.de`, containing your full name and address, so that we have a rough idea of the number of users. We also hope that this number will be large enough to convince various agencies that GAP is a project worthy of (financial) support. If you publish some result that was partly obtained using GAP, we would appreciate it if you would cite GAP, just as you would cite another paper that you used. Again we would appreciate if you could inform us about such a paper.

We distribute the **full source** for everything, the C code for the kernel, the GAP code for the library, and the LaTeX code for the manual,

which has at present about 800 pages. So it should be no problem to get GAP, even if you have a rather uncommon system. Of course, ports to non UNIX systems may require some work. We already have ports for IBM PC compatibles with an Intel 80386 or 80486, for the Apple Macintosh under MPW, and for the Atari ST. We also hope to provide a standalone port for the Apple Macintosh in the near future (there is already such a port of GAP 3.1). Note that about 4 MByte of main memory and a harddisk are required to run GAP.

GAP 3.2 (currently – May 1993 – at patchlevel 2) can be obtained from several **ftp** servers, including:

`samson.math.rwth-aachen.de`
: Lehrstuhl D fur Mathematik, RWTH Aachen, Germany;
directory `/pub/gap/`.

`ftp` to that server, login as user `ftp` and give your full e-mail address as password. GAP is in the directory `pub/gap`. Then get the file `README`, which contains further instructions on how to get and install GAP.

For users in the Eastern countries `ftp` may not be a viable option. They should contact Victor Ufnarovski (`91csjmol@math.moldova.su`), who has offered to distribute GAP to such users.

6 The Future of GAP

> See ye not all these things?
> Verily I say unto you,
> there shall not be left here
> one stone upon another,
> that shall not be thrown down.
> (Matthew 24:2)

Clearly GAP will contain bugs, as any system of this size, though currently we know none. Also there are things that we feel are still missing,

and that we would like to include into GAP. We will continue to improve and extend GAP. We will release new versions quite regulary now, and about three or four upgrades a year are planned. Make sure to get these, since they will in particular contain bug-fixes.

We are committed however, to staying upward compatible from now on in future releases. That means that everything that works now will also work in those future releases. This is different from the quite radical step from GAP 2.4 to GAP 3.1, in which almost everything was changed.

Of course, we have ideas about what we want to have in future versions of GAP. However we are also looking forward to your comments or suggestions.

7 The GAP Forum

We have also established a GAP forum, where interested users can discuss GAP related topics by e-mail. In particular this forum is for questions about GAP, general comments, bug reports, and maybe bug fixes. We, the developers of GAP, will read this forum and answer questions and comments, and distribute bug fixes. Of course others are also invited to answer questions, etc. We will also announce future releases of GAP on this forum. So in order to be informed about bugs and their fixes as well as about additions to GAP we recommend that you subscribe to the GAP forum.

Send an e-mail message to `listserv@samson.math.rwth-aachen.de` to subcribe to the GAP forum. This message should contain the line `subscribe gap-forum` *your-name*, where *your-name* should be your full name, not your e-mail address. You will receive an acknowledgement, and from then on all e-mail messages sent to the e-mail address `gap-forum@samson.math.rwth-aachen.de`.

`listserv@samson.math.rwth-aachen.de` also accepts the following requests: `help` for a short help on how to use `listserv`, `unsubscribe`

`gap-forum` to unsubscribe, `recipients gap-forum` to get a list of subscribers, and `statistics gap-forum` to see how many e-mail messages each subscriber has sent so far.

If you have technical problems or problems with the installation of GAP, write to `gap-trouble@math.rwth-aachen.de` instead of the GAP forum, as such discussions are usually not very interesting for a larger audience. Your e-mail message will be read by several people here, and we shall try to provide support.

Martin Schönert
Lehrstuhl D für Mathematik
RWTH
Templergraben 64,
5100 Aachen,
Germany
e-mail *m.schoenert@math.rwth − aachen.de*

Received June 22, 1993