An Invitation to Computational Group Theory

J. Neubüser

Introduction

Throughout the second week of the Groups '93 meeting a workshop on Computational Group Theory (CGT for short) took place. The underlying mathematical methods were described in four series of lectures ('Finitely Presented Groups', 'Collection Methods', 'Permutation Groups', 'Representations') and three single lectures ('Cohomology Groups', 'Matrix Groups', 'Groups, Graphs and Designs'). The practical aspect was present in a lecture 'Introduction to GAP', a course 'Programming in GAP' and exercises using GAP during the afternoons. It is a pleasure to thank the large number of colleagues who helped running this workshop.

During the first week, I gave a plenary talk which had the same title as this paper. It had the dual function to give information on the plan and intention of the workshop, but also in a wider sense to discuss the present state of CGT and its role in group theory at large. In parts the talk reflected rather personal viewpoints. Since in that it differs from papers normally read in a conference, I had doubts, if it should at all go into the proceedings. The editors have encouraged me to write it up, so here it is, much in the form of the talk delivered.

CGT may be described as comprising the development, analysis, implementation and use of group theoretical algorithms. The first two depend on and draw from the corresponding part of group theory and hence teaching the methods of CGT is best organized following the lines of the theory - it definitely was organized that way in our workshop. However, a good deal of the strength of CGT for application to problems in present day group theoretical research stems from the fact that in the presently available CGT program systems such as Cayley [BC91] or GAP [S^+93] a broad variety of methods is at hand - such as it is in a researcher's mind. I want to start by illustrating this point by a case study taken from the Aachen Diplom thesis of Alexander Hulpke [Hul93] which obtains a rather detailed analysis of a fairly big group using just standard methods, in this case in GAP - and, of course, the mathematical insight into the relevant theory and the available tools. In a second section I want to sketch some history and some highlights of CGT which will naturally lead to the last section with a - in parts critical - discussion of the present state and problems of the field.

1. A case study.

The main task of this Diplom thesis was the detailed analysis and re-implementation in GAP of an algorithm for the determination of the character table of a finite group from its class multiplication coefficients, a method that had been proposed by J. D. Dixon [Dix67] and improved and implemented in Cayley by G. Schneider [Sch90]. Originally just to test this implementation, some hitherto unknown character tables of maximal subgroups of sporadic simple groups were determined in the thesis. One of these is a soluble maximal subgroup of order 3 265 173 504 in Fi_{23}. This table has in fact been used meanwhile for the determination of the 5-modular character table of Fi_{23} by G. Hiss.

I shall describe an analysis of this subgroup, listing and commenting the GAP commands used, in order to demonstrate the ease of using such a program system. At certain stages printing of a result of a computation is suppressed by a ';;'. Also at some stages the GAP function `time` is called which prints the CPU time used by the last command (in ms) in order to give an idea how long such an computation needs (on a DEC station).

The Cambridge ATLAS [CCN+85] provides a list of maximal subgroups of Fi_{23}, which for the first class of maximal subgroups contains:

Order	Index	Structure	Character
129 123 503 308 800	31 671	$2 \cdot Fi_{22}$	1a+782a+30 888a

This tells us that Fi_{23} has a (faithful) transitive permutation representation of degree 31 671, the character of which is the sum of three irreducibles 1a, 782a, and 30 888a. The corresponding line for our subgroup, called S in the sequel, contains

Order	Index	Structure	
3 265 173 504	1 252 451 200	$3_+^{1+8} \cdot 2_-^{1+6} \cdot 3_+^{1+2} \cdot 2S_4$	N(3B)

giving us the information that it can be obtained as normalizer of an element of class $3B$. In addition to this information we use:

- a generating set (of two elements) for Fi_{23} in a permutation representation of degree 31 671, which we got from R. A. Wilson,
- the character table of Fi_{23}, available in the GAP library of character tables, which among many others contains all character tables of the ATLAS.

We start by reading in this character table and determining the permutation character of degree 31 671 as the sum of those three irreducibles, which (as in the ATLAS) have numbers 1, 2, and 6 in the list of characters:

```
gap> c:=CharTable("Fi23");;
gap> permchar:=Sum(c.irreducibles{[1,2,6]});;
gap> permchar[1];
31671
```

Fi_{23} has four classes of elements of order 3. The following commands tell us that these four classes have numbers 5, 6, 7, and 8 in the character table and that they can be discriminated by the value of the permutation character:

```
gap> Filtered([1..98],i->c.orders[i]=3);
[ 5, 6, 7, 8 ]
gap> permchar{last};
[ 351, 324, 135, 27 ]
```

So elements in class $3B$ (the second class of elements of order 3) have 324 fixed points.

The table head of the character table of Fi_{23}, either in the ATLAS or from the GAP library of character tables, gives us information about our chances of finding an element from class $3B$ in a random search. The centralizer of an element of class $3B$ is of order 1 632 586 752, that is, we have a probability of $1 : 1\,632\,586\,752$ that a random element of Fi_{23} is in class $3B$. However from the power map we see that elements of class $3B$ can be obtained i.a. as powers of elements of classes 36A, 36B, and 27A, which are selfcentralizing, so that we have an almost 10 % chance to hit an element of one of these 3 classes.

The following GAP routine (which uses that classes in the character table of Fi_{23} are ordered by increasing order of elements) tells us that altogether we have more than a 16 % chance to find an element with a power in class $3B$ by a random search.

```
gap> roots:=[6];;
>    for i in [1..Length(c.classes)] do
>      if ForAny(c.powermap,j->j[i] in roots) then
>        AddSet(roots,i);
>      fi;
>    od;
gap> roots;
[ 6, 15, 17, 19, 21, 33, 34, 35, 36, 43, 45, 46, 47, 52, 66,
 67, 68, 69, 70, 71, 72, 83, 87, 93, 94 ]
gap> Sum(roots,i->1/c.centralizers[i]);
257647/1594323
```

We prepare the search for such elements in $G := Fi_{23}$ by calculating a stabilizer chain

$$G = G_0 > G_1 > \ldots > G_n = \langle 1 \rangle,$$

where $G_i = Stab_{G_{i-1}}(i)$, and coset representatives of G_i in G_{i-1} for each $i = 1, \ldots, n$. Since we know the order of Fi_{23} we can supply it to the function `StabChain`, which can use it as a stopping criterion for random methods which is much faster for such big groups than the deterministic Schreier-Sims Algorithm.

```
gap> Read("fischer23.grp");           # taken from a local file
gap> StabChain(Fi23,rec(size:=4089470473293004800,
> random:=100));;time;
154834
```

The `time` function tells us that this took about two and a half minutes.

We can now use a GAP function `Random` which chooses for each i a random coset representative of G_i in G_{i-1} and multiplies these. We search for elements that have powers in class $3B$, using the test if the number of fixed points is indeed 324, by the following self-explanatory GAP routine.

```
gap> opdom:=PermGroupOps.MovedPoints(Fi23);;
gap> found:=false;; g:=();; h:=();;
gap> repeat
>      g:=Random(Fi23);
>      if Order(Fi23,g) mod 3 = 0 then
>        h:=g^(Order(Fi23,g)/3);
>        if Number(opdom, i->i^h=i) = 324 then
>          found:=true;
>        fi;
>      fi;
>    until found;time;
41931
```

When this stops, we check that the element h is indeed of order 3 and has 324 fixed points. We also note, that we first found a root g of h and not directly an element of order 3, as we had already suspected.

```
gap> Order(Fi23,h);Number(opdom,i->i^h=i);
3
324
gap> Order(Fi23,g);
27
```

We now set up a subgroup H generated by h and call a GAP function to compute its normalizer in Fi_{23}. We see that finding the normalizer does take about 21 minutes.

```
gap> H:=Subgroup(Fi23, [h]);;
gap> N:=Normalizer(Fi23, H);;time;Size(N);
1263599
3265173504
```

So we now have our maximal subgroup of order 3 265 173 504 as a permutation group N of degree 31 671. It would be advantageous to have a faithful permutation representation of this group of smaller degree. As a simple attempt to go in this direction, we determine the orbits of N:

```
gap> orb:=Orbits(N,opdom);;List(orb,Length);
[ 11664, 19683, 324 ]
```

The representation on the orbit of length 324 cannot be faithful since it consists of fixed points of h, however the representation on the second largest orbit is faithful:

```
gap> P:=Operation(N,orb[1]);;
gap> MakeStabChainRandom(P);;time;Size(P);
73432
3265173504
```

However since we know S to be soluble, we prefer to work with a polycyclic presentation, called an AG-group by tradition [LNS84]. We obtain this by

```
gap> A:=AgGroup(P);;time;Size(A);
1101031
3265173504
gap> A.name:="Fi23M7";;
```

in about 18 minutes. Again for safety (see section 3 of this paper) we have checked the order. We may now get information such as the derived series in rather short time (5 seconds):

```
gap> ds:=DerivedSeries(A);;time;Length(ds);
4953
11
gap> List(ds,i->Collected(Factors(Size(i))));
[ [ [ 2, 11 ], [ 3, 13 ] ], [ [ 2, 10 ], [ 3, 13 ] ],
  [ [ 2, 10 ], [ 3, 12 ] ], [ [ 2, 8 ], [ 3, 12 ] ],
  [ [ 2, 7 ], [ 3, 12 ] ],
  [ [ 2, 7 ], [ 3, 10 ] ], [ [ 2, 7 ], [ 3, 9 ] ],
  [ [ 2, 1 ], [ 3, 9 ] ],
  [ [ 3, 9 ] ], [ [ 3, 1 ] ], [ [ 1, 1 ] ] ]
```

This confirms some of the information about the structure of the group quoted above from the ATLAS. However using methods described e.g. in [CNW90] we can obtain further information about the splitting of extensions, e.g. we can determine representatives of the conjugacy classes of complements of the second last group in the derived series which is an extra special group of order 3^9.

```
gap> compcl:=List([1..8],i->Complementclasses(ds[i],ds[9]));;
gap> List(compcl,Length);
[ 3, 3, 1, 1, 9, 3, 1, 1 ]
```

We can also determine (in less than a minute) the conjugacy classes of elements using methods from [MN89]

```
gap> conjcl:=ConjugacyClasses(A);;time;Length(conjcl);
49309
181
```

We see that there are 181 classes. Finally we can determine the character table of this group, using Alexander's implementation of the Dixon/Schneider method.

```
gap> C:=CharTable(A);;time;
18365368
```

This is indeed the longest computation of the whole run, taking a little over 5 hours. Printing a 181×181 table is not what we want, but we can use GAP to extract information from the character table. We may for instance ask for the maximal degree of an irreducible character.

```
gap> Maximum(List(C.irreducibles,i->i[1]));
18432
```

Or we may ask for the extension of the rationals generated by the character values:

```
gap> irrat:=Filtered(Union(C.irreducibles),i->not IsRat(i));
[ -4*E(8)-4*E(8)^3, -2*E(8)-2*E(8)^3, -E(8)-E(8)^3,
  E(8)+E(8)^3, 2*E(8)+2*E(8)^3, 4*E(8)+4*E(8)^3 ]
gap> alpha:=irrat[4];
E(8)+E(8)^3
gap> X(Rationals).name:="x";;
gap> Polynomial(Rationals,MinPol(alpha));
x^2 + 2
gap> Polynomial(Rationals,MinPol(E(8)));
x^4 + 1
```

We may also ask for the kernel of a character and in this particular case identify it with the third last group in the derived series.

```
gap> char144:=First(C.irreducibles,i->i[1]=144);;
gap> KernelChar(char144);
[ 1, 2, 3, 4, 5, 6, 7, 8 ]
gap> Sum(last,i->c.classes[i]);
39366
gap> List(ConjugacyClasses(A){last2},
>           x->x.representative in ds[8]);
[ true, true, true, true, true, true, true, true ]
```

2. Some history and highlights

As we have seen, the case study described in the last section can be performed

- in a day or two (with only a few hours in attention; the character table can be done overnight)
- on standard equipment (a UNIX workstation in this case, a PC under DOS would do as well)
- using standard methods, available in CGT program systems (GAP in this case).

In this section I want to outline some steps in the development of CGT that led to this situation, not going into much detail.

Prehistory (before 1953). Not only did group theory get started by the very computational problem of the solvability of quintics by radicals but also (hand-) computation of groups from Mathieu's discovery of the first sporadics to Hölder's determination of the groups of order pqr made up a good deal of last century's group theory. However Dehn's formulation of word, conjugacy, and isomorphism problem for finitely presented groups in 1911 [Deh11] (even though it had precursors) may be thought of as the beginning of the prehistory proper of CGT, focussing attention on the request for group theoretical algorithms. Two points are worth noting:

- The challenge came from topology, not from inside group theory, and indeed even now people using groups outside group theory are strong "customers" of CGT.

- Novikov's proof of the algorithmic unsolvability of the word problem put an end to the hope that group theory could fulfil Dehn's request. Soon after came proofs of the non-existence of algorithms that could **decide** if a finitely presented group is trivial, finite, abelian, etc. (see G. Baumslag's book [Bau93] for a vivid description).

Nevertheless in 1936 J. A. Todd and H.S.M. Coxeter [TC36] provided at least a systematic method of attempt for showing finiteness by "coset enumeration" and today CGT provides a whole bunch of methods for the investigation of finitely presented groups that in frequent use have proved quite powerful, even though they are not decision algorithms.

Of the period predating the use of real computers I want to mention three more events:

- In 1948 H. Zassenhaus described an algorithm for the classification of space groups [Zas48] that much later has been put to very practical use.

- In 1951 M. H. A. Newman in a talk at the 'Manchester University Computer Inaugural Conference' [New51] proposed to use probabilistic methods for getting some insight into the vast number of groups of order 256. The title of the talk: "The influence of automatic computers on mathematical methods" is remarkable for that time, and the fact that the (56 092) groups of that order were only determined in 1989 by E. A. O'Brien [O'B91] should be noted.

- Perhaps most notable because of its foresight however is a quotation from a proposal of A. Turing in 1945 to build an electronic computer: "There will positively be no internal alterations to be made even if we wish suddenly to switch from calculating the energy levels of the neon atom to the enumeration of groups of order 720" (see [Hod85], page 293).

Early history (1953–1967) of CGT may be thought of as starting with the first implementations of computing methods for groups. The first, about 1953, that I am aware of, are a partial implementation of the Todd-Coxeter method by B. Haselgrove on the EDSAC II in Cambridge (see [Lee63]) and of methods for the calculation of characters of symmetric groups by S. Comet on the BARK computer in Stockholm [Com54]. Other areas of group theory were tried soon after. E.T. Parker and P.J. Nicolai made an - unsuccessful - search for analogues of the Mathieu groups [PN58], 1959 programs for calculating the subgroup lattice of permutation groups [Neu60] and a little later for polycyclicly presented 2-groups were written. Other methods and special investigations followed, programs of this time were written mostly in machine code, use of all kinds of trickery to save storage space and (although not quite as urgently) computing time was crucial. The end of this period, in which CGT started to unfold but had hardly contributed results to group theory that would greatly impress group theorists, is roughly marked by the Oxford conference "Computational Problems in Abstract Algebra" in 1967 [Lee70]. Its proceedings contain a survey of what had been tried until then [Neu70] but also some papers that lead into the

Decade of discoveries (1967–1977). At the Oxford conference some of those computational methods were presented for the first time that are now, in some cases varied and improved, work horses of CGT systems: Sims' methods for handling big permutation groups [Sim70], the Knuth-Bendix method for attempting to construct a rewrite system from a presentation [KB70], variations of the Todd-Coxeter method for the determination of presentations of subgroups [Men70]. Others, like J. D. Dixon's method for the determination of the character table [Dix67], the p-Nilpotent-Quotient method of I. D. Macdonald [Mac74] and the Reidemeister-Schreier method of G. Havas [Hav74] for subgroup presentations were published within a few years from that conference.

However at least equally important for making group theorists aware of CGT were a number of applications of computational methods. I mention three of them: The proof of the existence of Lyons' sporadic simple group by C. C. Sims in 1973, using his permutation group methods [Sim73], the determination of the Burnside group $B(4,4)$ of order 2^{422} by M. F. Newman and G. Havas using an extension of the original p-Nilpotent-Quotient method

[New76], and the determination of the (4783 isomorphism classes of) space groups of 4-dimensional space [BBN+78], using not only Zassenhaus' algorithm but also the possibility to find all subgroups of the maximal finite subgroups of $GL(4, \mathbf{Z})$ using the programs for the determination of subgroup lattices.

This progress encouraged attempts to design CGT systems in which various methods could be used without having to translate data from one program to the other. By 1974 a first "Aachen – Sydney Group System" was operational [Can74], and in 1976 John Cannon published "A Draft Description of a Group Theory Language Cayley" [Can76], which may be thought of as a turn to

Modern Times. The claim that since about 1977 the development of CGT is speeding up rapidly can be justified by looking at four aspects: results, methods, systems and publicity.

Results. Concrete computational results, of which again I list only some: Sims proved the existence of the Babymonster of order 4 154 781 481 226 426 191 177 580 544 000 000 as a permutation group of degree 13 571 955 000 using very special implementations based on his permutation group techniques [Sim80], the existence of Janko's J_4 was proved using computational techniques for modular representations [Nor80]. p-Nilpotent Quotient techniques are now strong enough to show e. g. that the class 18 factor of the restricted Burnside group $R(2,7)$ has order 7^{6366} [HNV-L90]. Following a proposal of M. F. Newman [New77], E. A. O'Brien implemented a p-group generation program sufficient to determine the (58760 isomorphism classes of) groups of order 2^n, $n \leq 8$ [O'B91]. Making and checking the Cambridge "Atlas of Finite Groups" [CCN+85], probably the most widely used group theoretical table, involved a great deal of implementation and use of group theoretical programs (in particular for working with characters), and the same holds for books listing Brauer trees of sporadics [HL89] or perfect groups [HP89] as well as for other listings, e. g. of primitive or transitive permutation groups.

Methods. These and many more concrete computations were made possible by a large number of new methods and their integration into general and specialized systems.

I have mentioned already the p-group generation method, which builds on the p-Nilpotent Quotient Algorithm (pNQ) and links to the recent exploration of the possibility to classify families of p-groups of constant coclass with the help of space groups [L-GN80]. Another very recent offspring of the pNQ is a (proper) Nilpotent Quotient Algorithm stepping down the lower central series of a finitely presented group. A number of proposals have been made and a couple of them implemented recently for finding soluble factorgroups of finitely presented groups, using concepts such as cohomology groups, modular representations, or Gröbner bases [L-G84],[Ple87],[Sim90].

Working via homomorphic images has become the method of choice for handling polycyclicly presented finite soluble groups [LNS84] but also has become indispensable for handling permutation groups [KL90].

The methods for the investigation of permutation groups have almost undergone a revolution, bringing in structure theory such as the O'Nan-Scott classification of primitive groups or even the classification of finite simple groups, now e. g. allowing the determination of the composition series of groups of degrees in some cases up into the hundred thousands. It is particularly interesting to note that some of these new methods for permutation groups, which have now become very practical, too, first were brought in through rather theoretical discussions of the complexity of permutation group algorithms. ([Neu86],[Kan85],[KT88],[BLS88],[BCF$^+$ar], [BCFS91],[BS92], to mention just a small selection of many papers on this subject.)

A broad variety of methods, many interactive, are available for working with representations and characters [NPP84],[LP91].

A long neglected, but now very rapidly growing branch of CGT are methods for the study of matrix groups over finite fields [NP92], making strong use of the Aschbacher classification [Asc84].

Systems. A program system comprises various components: storage management, a problem-oriented language for both interactive use and for writing programs, that can call system functions (and possibly directly access data), a library of functions that can be applied to the objects studied, and libraries of such objects. The first general system for CGT, developed in the mid-70-ties, was the Aachen-Sydney Group System, for which J. Cannon provided storage management (stackhandler) and language (Cayley). Functions were written in Fortran. This system developed into the Cayley System [BC91], the functions of which were semiautomatically translated to C about 1987. Cayley has been remodelled recently under the name MAGMA [CP93] and extended to a general Computer Algebra system reaching beyond CGT.

Another general system for CGT called GAP (**G**roups, **A**lgorithms, and **P**rogramming) was started in Aachen in 1986. Its design is influenced by the computer algebra system Maple: GAP has a kernel, written in C, containing storage management, language interpreter (in the future also a compiler) for the GAP language and basic time-critical functions. The large majority of the GAP functions is written in the GAP language and so can be understood, checked and altered, a point on which I want to comment in the last section.

Both Cayley and GAP have found wide use. With Cayley more exact figures can be given, since it must be licensed for a certain fee, J. Cannon reports over 200 licenses by 1990 [Can91]. GAP can be obtained free of charge via ftp and handed on so that no full control of its spread exists. An indication are the about 240 members of the GAP-forum and more than 350 reports of installation that we obtained.

During the late 70-ties and the 80-ties a number of very specialized systems have been written which became absorbed or outdated by Cayley and GAP, some others complement the scopes of the two general systems, among them Quotpic [HR93], giving a very good visualisation for the steps in the calculation of factor groups of finitely presented groups, MOLGEN [GKL92] for the application of group theoretical methods to the construction of graphs, in particular representing the structure of organic compounds, and MOC [LP91] for the construction of modular character tables.

Publicity. Already the demand for CGT systems indicates the widespread use of algorithmic methods in research (and increasingly also teaching) of group theory. The presence of CGT in general meetings on group theory, such as all four Groups St. Andrews meetings 1981, 1985, 1989, and now 1993, as well as the growing frequency of specialized meetings on CGT are further indications. While the first meeting fully devoted to CGT was held in Durham 1982 [Atk84], further ones were Oberwolfach 1988 and 1992, DIMACS 1991 [FK93] and 1994. In addition to a number of surveys (see in particular [CH92] for a recent one with many references) two monographs on parts of the field have appeared recently. While Greg Butler [But91] tries to introduce Computer science students with no preknowledge of group theory to computational methods for permutation groups, Charles Sims [Sim94] gives an authoritative account of methods for the investigation of finitely presented groups, emphasizing common features of various approaches.

3. Some concerns.

Summing up what has been reported in the preceding sections, one may come to very optimistic conclusions. Computational Group Theory has provided evidence of its power, many of its methods are generally and comfortably available and widely used. And for the future: Computers are getting faster, storage space bigger, both cheaper, methods better, program systems more comprehensive and easier to use. So the future is all gleaming with promise! Is it? I have some concerns.

(i) I mentioned 1967, the year of the Oxford Conference, as the time of breakthrough for CGT. That same year Huppert's book "Endliche Gruppen I" appeared. It is still on my shelf and as useful as 27 years ago, while none of the programs that we proudly talked about in Oxford is running any more. Of course Huppert built on more than hundred years of research in Group Theory and almost hundred years experience of writing books about it, while Computational Group Theory and the art of implementing its methods were still in their infancy. However, I have to admit: While Huppert's book will most likely still be on your shelves in 2021, after the next 27 years, one must have serious doubts if you will still be able to use GAP (or MAGMA) then. Computers and computer languages are still changing rapidly. It is, for instance, very much in vogue to bet on parallel computers as the tool of the future, but it is by no means clear to me which of the many models of parallel computers and operating systems and languages for them will make it. And we are still lacking safe methods to preserve the huge amount of work that has gone and is going into the development of systems such as GAP or MAGMA for even a foreseeable time span. We do not even have defined standards to save the mathematical facts – character tables, group classifications etc. – that have already been created by use of CGT in a way that guarantees for many years the possibility to use and to check them.

(ii) I had the privilege of getting involved with the proofreading of Huppert's book. Almost every page has been read, rewritten and reread several times over with immense care. Nevertheless the last edition of the book has almost three pages of errata. I have to admit for GAP that we did not have the manpower and time to do checking of any comparable intensity and I doubt that the situation is much better with other systems. But worse: On page 128 Huppert's book refers to the Schreier conjecture as to the "Schreibweise Vermutung". A typo like that in a textbook causes at best a smile by the reader, in a program it may cause some unpredictable action of the computer.

To make one point clear: This is not reviving the old prejudices that one cannot trust results obtained by computer calculations. Modern computers are by orders of magnitude more reliable in doing computations by rules than human brains. When we talk about bugs we talk about human mistakes made in setting up these rules (i. e. programs) which are of exactly the same nature as mistakes that occur in proofs. In program systems such as GAP with presently about 50 000 lines of C-code in the kernel, 120 000 lines of GAP-code in the program library, and a manual of about 1000 pages, bugs are practically unavoidable. Among these, those that cause the system to crash or produce obvious nonsense are annoying, the real dangerous ones are those that produce wrong output that still looks possible or even plausible at first sight. What can be done about this by system developers?

Programs have always been tested by running examples, it is a problem of mathematical insight and imagination to choose a sufficiently representative set of such test examples and a problem of manpower to choose it big enough. We will come back to this with point (iv).

Use of modern programming languages that allow or even enforce more transparent implementation of algorithms has certainly helped very much to avoid and to find bugs – in the same way as modern standards of formalizing and formulating proofs have done this with writing mathematics. However it does not totally eliminate the problem – again the same has to be said of standards of writing proofs.

Finally on this topic: Providing corrections to a book is an easy matter: I use my 1967 copy of Huppert's book with a copy of those 3 pages of errata; if I want, I just need a pencil to mark them on the margin. What about removing bugs from a program? If source code is interpreted, it is still reasonably easy to apply a patch. If a (compiled) executable is used, either, if available, the source must be corrected and then recompiled or, if not, a new executable must be gotten and installed. Each is a much more cumbersome procedure that needs much more assistance from the side of the system developer.

(iii) You can read Sylow's Theorem and its proof in Huppert's book in the library without even buying the book and then you can use Sylow's Theorem for the rest of your life free of charge, but – and for understandable reasons of getting funds for the maintenance, the necessity of which I have pointed out in (i) and (ii) – for many computer algebra systems license fees have to be paid regularly for the total time of their use. In order to protect what you pay for, you do not get the source, but only an executable, i. e. a black box. You can press buttons and you get answers in the same way as you get the bright pictures from your television set but you cannot control how they were made in either case.

With this situation two of the most basic rules of conduct in mathematics are violated: In mathematics information is passed on free of charge and everything is laid open for checking. Not applying these rules to computer algebra systems that are made for mathematical research (as is the case practically exclusively with systems for CGT) means moving in a most undesirable direction. Most important: Can we expect somebody to believe a result of a program that he is not allowed to see? Moreover: Do we really want to charge colleagues in Moldava several years of their salary for a computer algebra system? And even: If O'Nan and Scott would have to pay a license fee for using an implementation of their ideas about primitive groups, should not they in turn be entitled to charge a license fee for using their ideas in the implementation?

(iv) The preceding discussion describes a dilemma. On one hand CGT systems are as useful, in fact even more indispensible, for giving algorithmic ideas an impact on the progress of our knowledge on concrete groups as are books for the dissemination of theoretical insight. On the other hand it should have become clear that development and maintenance of such systems pose more problems and involve more continuously needed manpower than writing a book. I hope also to have given good reason, why I do not think that license fees are a desirable solution. Most systems have obtained some support through research grants, – in the case of GAP we gratefully acknowledge such support by the Deutsche Forschungsgemeinschaft – but typically such support is given during a restricted time for original development rather than continuously for maintenance.

I do not have a patent remedy for the problem but let me close by describing first what we try with GAP and then what in my view is needed on a broader scale if CGT is to continue to develop as well as it has since Oxford 1967.

(v) Our policy: Both, (C-) kernel and GAP library of functions are distributed with full source, free of charge, through anonymous ftp. We provide patches at intervals of a few months and release new versions of the system about every 9–12 months. Some C-programs written by other teams, devoted to special problems and tuned for these to high efficiency, are linked to GAP as 'share libraries' that can be called from GAP but remain under the responsibility of their authors.

We maintain an electronic 'GAP-forum' not only for discussion of the use of GAP in research and teaching but also (and at least as important) for bug reports of users that are thus made generally known immediately, and we encourage users to join the forum. In addition, being helped by some experienced users, we try to provide advice with technical problems, e. g. installation problems, that are sent to an address 'GAP-trouble'.

The kernel of GAP contains time-critical parts of the system, such as storage management, language interpreter and basic functions. It is clear that most users of GAP neither know nor want to know much about the methods used in the kernel, they rightly just want to rely on them. Therefore the kernel is kept as small as possible so that its development and maintenance can be managed by very few people who are highly experienced with system building (at present in the very first place Martin Schönert). On the other hand the fact that the much larger GAP library is written in the more transparent GAP language allows users to play an active part in the development of GAP. They can and should – as always when using a program – exercise all their expertise to check critically if results "look right", if they have doubts they can look at the code, trying to locate or even to correct mistakes, but in any case they can and should relate their doubts through the GAP-forum to the whole community of users and to the developers of the system. In fact this scheme has worked quite well during the last few years and has helped considerably improving the reliability of GAP.

It goes almost without saying that the relative ease of reading and writing GAP language also gives users a much better chance to adapt an existing function or to write a new one for their particular problem and in fact this is also done frequently now. It would be most desirable, however, if in many more cases some extra effort would be spent in preparing such "private" programs for general use and making them available to the public. Again with GAP we try to provide some organisational help for such publicizing of programs. Of course the ability to control, adapt, or amend functions in GAP presupposes at least some basic knowledge about group theoretical algorithms, which should therefore start to enter group theory courses, not at all to replace but to complement some parts of the theory.

(vi) I have emphasized the central role that we attribute to cooperation in our 'GAP policy'. There still remains a large amount of work to be done by rather few who develop and maintain GAP. It is important to realize that such work, if it is to live up to the state of the art, must be performed in close contact with the progress of group theory and with group theoretical problems that are presently studied. Progress in CGT in scope but also in efficiency of implementations has come far more from a better understanding of the underlying mathematics than from better implementation techniques. (These rather have their importance for reliability, flexibility, and portability of the code.) That is, the development of a group theory system is a job for group theorists (some of whom are also good at system programming) rather than for professional system programmers. But then it must also be realized that the work of such people must be recognized as contributing to mathematics. I still often enough see papers by authors who for this paper have successfully used GAP (or MAGMA or some other systems) and refer to this use by saying: "...and then by computer calculation we obtained...". Would you quote a theorem from a paper that has an author and a title by saying: "... and then by browsing through our library we became aware of the following fact ..."? Of course if we ask to change this habit, on the part of the system developers we have to be more careful in attributing contributions to individuals (with GAP we try at least) and also we have to find ways of certifying contributions to systems similar to established methods with publishing papers.

Summing up, in my view it is necessary to avoid everything that would separate work on the design **and** implementation of algorithms from other mathematical work. Rather we have to make every possible effort to adjust to habits and to adopt rules of conduct that are common practice in other parts of mathematics. For this Computational Group Theory needs the closest cooperation with all other parts of Group Theory.

It is this that the title of the paper wants to express: I want to invite you to Computational Group Theory, which should not be considered as a group theoretical fool's paradise where shiny black boxes spit out character tables and cohomology groups, rather it should be considered as a field that needs a lot of tending, but is also worth your help in tending it.

References

[Asc84] M. Aschbacher, On the maximal subgroups of the finite classical groups, *Invent. Math.* **76** (1984), 469–514.

[Atk84] M. D. Atkinson, editor. *Computational Group Theory, Durham, 1982.* Academic Press, 1984.

[Bau93] G. Baumslag. *Topics in Combinatorial Group Theory.* Lectures in Mathematics, ETH Zürich. Birkhäuser, 1993.

[BBN+78] H. Brown, R. Bülow, J. Neubüser, H. Wondratschek, and H. Zassenhaus. *Crystallographic groups of four-dimensional space.* Wiley-Interscience, New York, 1978.

[BC91] W. Bosma and J. Cannon. *A handbook of Cayley functions.* Computer Algebra Group, Sydney, Australia, 1991.

[BCF+ar] L. Babai, G. Cooperman, L. Finkelstein, E. Luks, and A. Seress, Fast monte carlo algorithms for permutation groups, *J. Comp. Syst. Sci.* (to appear).

[BCFS91] L. Babai, G. Cooperman, L. Finkelstein, and A. Seress. Nearly linear time algorithms for permutation groups with a small base. In *Proc. International Symposium on Symbolic and Algebraic Computation, Bonn (ISSAC '91)*, pages 200–209, 1991.

[BLS88] L. Babai, E. Luks, and A. Seress. Fast management of permutation groups. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 272–282, 1988.

[BS92] R. Beals and A. Seress. Computing composition factors of small base groups in almost linear time. In *Proc. 24th ACM Symp. on the Theory of Computing (1992)*, pages 116–125, 1992.

[But91] G. Butler. *Fundamental algorithms for permutation groups*, volume 559 of *Lecture Notes in Computer Science.* Springer, Berlin, 1991.

[Can74] J. Cannon. A general purpose group theory program. In Newman [New74], pages 204–217.

[Can76] J. Cannon. A draft description of the group theory language Cayley. In Jenks [Jen76], pages 66–84.

[Can91] J. Cannon, A bibliography of Cayley citations, *SIGSAM Bulletin* **25** (1991), 75–81.

[CCN+85] J. H. Conway, R. T. Curtis, S. P. Norton, R. A. Parker, and R. A. Wilson. *ATLAS of finite groups.* Oxford University Press, 1985.

[CH92] J. Cannon and G. Havas, Algorithms for groups, *The Australian computer journal* **27** (1992), 51–60.

[CNW90] F. Celler, J. Neubüser, and C. R. B. Wright, Some remarks on the computation of complements and normalizers in soluble groups, *Acta Applicandae Mathematicae* **21** (1990), 57–76.

[Com54] S. Comet. On the machine calculation of characters of the symmetric group. In M. Riesz, editor, *Tolfte Skandinaviska Matematikerkongressen*, pages 18–23, Lund, 1954. Hakan Ohlssons Boktryckeri, Lund.

[CP93] J. Cannon and C. Playoust. *An Introduction to MAGMA*. School of Mathematics and Statistics, University of Sydney, 1993.

[Deh11] M. Dehn, Über unendliche diskontinuierliche Gruppen, *Math. Ann.* **71** (1911), 116–144.

[Dix67] J. D. Dixon, High speed computation of group characters, *Numer. Math.* **10** (1967), 446–450.

[FK93] L. Finkelstein and W. M. Kantor, editors. *Groups and computation, Proc. DIMACS Workshop, October 1991.* AMS-ACM, 1993.

[GKL92] R. Grund, A. Kerber, and R. Laue, MOLGEN, ein Computeralgebrasystem für die Konstruktion molekularer Graphen, *Com. Math. Chem.* **27** (1992).

[Hav74] G. Havas. A Reidemeister-Schreier program. In Newman [New74], pages 347–356.

[HL89] G. Hiss and K. Lux. *Brauer trees of sporadic groups.* Oxford University Press, 1989.

[HNV-L90] G. Havas, M. F. Newman, and M. R. Vaughan-Lee, A nilpotent quotient algorithm for graded Lie rings, *J. Symbolic Computation* **9** (1990), no. 5/6, 653–664.

[Hod85] A. Hodges. *Alan Turing, The Enigma of Intelligence.* Unwin Paperbacks, 1985.

[HP89] D. F. Holt and W. Plesken. *Perfect groups.* Oxford University Press, 1989.

[HR93] D.F. Holt and S. Rees. A graphics system for displaying finite quotients of finitely presented groups. In Finkelstein and Kantor [FK93], pages 113–126.

[Hul93] A. Hulpke. Zur Berechnung von Charaktertafeln. Diplomarbeit, Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, 1993.

[Jen76] R. D. Jenks, editor. *SYMSAC 1976*, Yorktown Heights, NY, 1976. Association Computing Machinery, New York.

[Kan85] W. Kantor, Sylow's theorem in polynomial time, *J. Comput. Syst. Sci.* **30** (1985), 359–394.

[KB70] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In Leech [Lee70], pages 263–297.

[KL90] W. Kantor and E. M. Luks. Computing in quotient groups. In *Proceedings of the 22nd ACM Symposium on Theory of Computing, 1990*, pages 524–534, 1990.

[KT88] W. Kantor and D. Taylor, Polynomial-time versions of Sylow's theorem, *J. Algorithms* **9** (1988), 1–17.

[Lee63] J. Leech, Coset enumeration on digital computers, *Proc. Cambridge Philos. Soc.* **59** (1963), 257–267.

[Lee70] J. Leech, editor. *Computational Problems in Abstract Algebra, Oxford, 1967.* Pergamon Press, Oxford, 1970.

[L-G84] C. R. Leedham-Green. A soluble group algorithm. In Atkinson [Atk84], pages 85–101.

[L-GN80] C. R. Leedham-Green and M. F. Newman, Space groups and groups of prime-power order I, *Arch. Math.* **35** (1980), 193–202.

[LNS84] R. Laue, J. Neubüser, and U. Schoenwaelder. Algorithms for finite soluble groups and the SOGOS system. In Atkinson [Atk84], pages 105–135.

[LP91] K. Lux and H. Pahlings. Computational aspects of representation theory of finite groups. In G. O. Michler and C. M. Ringel, editors, *Representation Theory of Finite Groups and Finite-Dimensional Algebras*, volume 95 of *Progress in Mathematics*, pages 37–64. Birkhäuser, 1991.

[Mac74] I. D. Macdonald, A computer application to finite p-groups, *J. Australian Math. Society* **17** (1974), 102–112.

[Men70] N. S. Mendelsohn. Defining relations for subgroups of finite index of groups with a finite presentation. In Leech [Lee70], pages 43–44.

[MN89] M. Mecky and J. Neubüser, Some remarks on the computation of conjugacy classes of soluble groups, *Bulletin of the Australian Mathematical Society* **40** (1989), 281–292.

[Neu60] J. Neubüser, Untersuchungen des Untergruppenverbandes endlicher Gruppen auf einer programmgesteuerten elektronischen Dualmaschine, *Numer. Math.* **2** (1960), 280–292.

[Neu70] J. Neubüser. Investigations of groups on computers. In Leech [Lee70], pages 1–19.

[Neu86] P. M. Neumann. Some algorithms for computing with finite permutation groups. In E. F. Robertson and C. M. Campbell, editors, *Proceedings of Groups – St. Andrews 1985*, London Math. Soc. Lecture Note Ser. 121, pages 59–92. Cambridge University Press, 1986.

[New51] M. H. A. Newman. The influence of automatic computers on mathematical methods. In F. C. Williams, editor, *Manchester University Computer Inaugural Conference*, pages 13–15, Manchester, 1951. Tillotsons (Bolton) Ltd., Bolton, England.

[New74] M. F. Newman, editor. *Second International Conference on the Theory of Groups, Canberra, 1973*, Lecture Notes in Math., vol. 372. Springer, Berlin, 1974.

[New76] M. F. Newman. Calculating presentations for certain kinds of quotient groups. In Jenks [Jen76], pages 2–8.

[New77] M. F. Newman. Determination of groups of prime-power order. In R. A. Bryce, J. Cossey, and M. F. Newman, editors, *Group theory, Proc. Miniconf., Austral. Nat. Univ., Canberra, 1975*, Lecture Notes in Math., vol. 573, pages 73–84. Springer, Berlin, 1977.

[Nor80] S. Norton, The construction of J_4, *Proc. of Symp. in Pure Math.* **37** (1980), 271–277.

[NP92] P. M. Neumann and C. E. Praeger, A recognition algorithm for special linear groups, *Proc. Lond. Math. Soc.* **65** (1992), 555–603.

[NPP84] J. Neubüser, H. Pahlings, and W. Plesken. CAS; design and use of a system for the handling of characters of finite groups. In Atkinson [Atk84], pages 195–247.

[O'B91] E. A. O'Brien, The groups of order 256, *J. Algebra* **143** (1991), 219–235.

[Ple87] W. Plesken, Towards a soluble quotient algorithm, *J. Symbolic Computation* **4** (1987), 123–127.

[PN58] E. T. Parker and P. J. Nicolai, A search for analogues of Mathieu groups, *Math. Tables Aids Comput* **12** (1958), 38–43.

[S$^+$93] M. Schönert et al. *GAP 3.3.* Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, 1993.

[Sch90] G. J. A. Schneider, Dixon's character table algorithm revisited, *J. Symbolic Computation* **9** (1990), no. 5/6, 601–606.

[Sim70] C. C. Sims. Computational methods in the study of permutation groups. In Leech [Lee70], pages 169–183.

[Sim73] C. C. Sims. The existence and uniqueness of Lyons' group. In T. Gagen, M. P. Hale, and E. E. Shult, editors, *Finite groups '72, Proceedings of the Gainesville conf., Univ. of Florida, Gainesville, Florida, 1972*, pages 138–141. North-Holland, Amsterdam, 1973.

[Sim80] C. C. Sims. How to construct a baby monster. In M. J. Collins, editor, *Finite simple groups II, Proc. Symp., Durham, 1978*, pages 339–345. Academic Press, 1980.

[Sim90] C. C. Sims, Implementing the Baumslag-Cannonito-Miller polycyclic quotient algorithm, *J. Symbolic Computation* **9** (1990), no. 5/6, 707–723.

[Sim94] C. C. Sims. *Computation with finitely presented groups.* Cambridge University Press, 1994.

[TC36] J. A. Todd and H. S. M. Coxeter, A practical method for enumerating cosets of a finite abstract group, *Proc. Edinburgh Math. Society (2)* **2** (1936), 26–34.

[Zas48] H. Zassenhaus, Über einen Algorithmus zur Bestimmung der Raumgruppen, *Comment. Math. Helv.* **21** (1948), 117–141.