CAYLEY: GROUP THEORY BY COMPUTER

Patrick Fitzpatrick

INTRODUCTION

CAYLEY is a sophisticated programming language for working with algebraic structures. Its principal domain is in the area of group theory but it may also be applied to rings, fields, modules and vector spaces. Not only does it contain a large compendium of preprogrammed group theoretical algorithms but also it provides the user with the facility to write and develop new programs. Cayley has taken up residence on the VAX 11/785 at UCG and is therefore accessible to group theorists throughout the country via the Higher Education Authority's network HEANET.

The purpose of this note is to give the reader a brief introduction to the main elements of Cayley and to provide a few examples to illustrate the power of the language. We avoid detailed technical discussion of syntax and format and make no claim to be comprehensive; rather it is our aim to whet the reader's appetite for "hands on" experience. Complete information on the current version of Cayley may be found in [1] and its updates (see also [2]).

Definition and Manipulation of Groups

There are various ways to define a group in Cayley: using generators and relations, as a permutation group, and as a matrix group.

Example (a) The program segment

```
dih4 : free (a,b);
dih4.relations : a↑2 = b↑2 = (a*b)↑4 = 1;
```

Financial assistance from the Faculty of Arts, UCC, is gratefully acknowledged.

defines the dihedral group of order 8 giving it the name *dih4*.

(b) The program

```
g : permutation group(8);
g.generators : (1,2,3), (4,5,6), (1,3,8);
```

defines *g* as the subgroup of Sym(8) generated by the given three elements.

(c) To define the group of 2x2 matrices of determinant 1 over GF(3) and call it *sl23* we need:

```
k : field(3);
v : vector space (2,k);
sl23 : matrix group (v);
sl23.generators : x = (1,1 : 0,1), y = (1,0 : 1,1);
```

The generating matrices are

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Observe that each Cayley statement ends with a semicolon and that no distinction is made between upper and lower case letters (although they may of course be used for clarity in programming). Also note that the symbol ↑ may be ^ on some terminals.

Elements are defined and manipulated in the obvious ways. Thus if *u* and *v* are previously defined elements of some group then

```
u↑2 * v↑-1, (u*v↑3)↑-1   and   u↑v
```

represent the elements u^2v^{-1}, $(uv^3)^{-1}$ and u^v $(=v^{-1}uv)$ respect-

ively. Furthermore the instruction

h = <u,v>;

defines *h* as the subgroup generated by $\{u,v\}$. A lot of information about a group *g* and its subgroups may be obtained using the *PRINT* command:

PRINT order(g), exponent(g), classes(g);

for example.

STRUCTURED PROGRAMMING

Cayley is a high-level language which allows structured programming in a group theoretic context. Thus we can implement the following constructions:

(i)
IF expression
THEN statements
[ELSE statements]
END;

(ii)
FOR i = 1 to 100 DO
statements
END;

(iii)
FOR EACH x IN s DO
statements
END;

(iv)
WHILE expression DO
statements
END;

(v)
REPEAT
statements
WHILE expression;
END;

Here the term *expression* denotes a variable which has the logical (Boolean) value *true* or *false* and in each of (iii) - (v) the program continues as long as *expression* takes the value *true*. The constructions are all subject to the usual constraints regarding their use in combinations such as nested sequences. Other commands *GOTO*, *LOOP* and *BREAK* may be used to transfer control from one part of a program to another. The *GOTO* command has its usual meaning while *LOOP* forces a continuation of the next cycle of a loop without carrying out the remainder of the statements associated with it and *BREAK* transfers control to the next statement after the *END* of the loop. Use of these constructions is particularly important in minimising the execution time of a given program.

STANDARD FUNCTIONS

One of the aspects which makes Cayley such an attractive working environment for the group theorist is the large library of standard functions available. Many of the group theoretic concepts that appear in everyday use are included: normalizer, centralizer, normal closure, core, conjugacy classes, generators, Sylow p-subgroup, upper and lower central series, derived and Frattini series, orbit, block, stabilizer for example. An expression of the form

normalizer(G,H)

represents the group $N_G(H)$ and can be used as it stands in a program. Some of the standard functions are restricted to groups defined in certain ways, but even these limitations can often be overcome by judicious programming.

In addition to the standard functions represented by keywords there are also built into Cayley several standard group theoretical algorithms for working with finitely presented groups: for instance, the Todd-Coxeter, Nilpotent Quotient and Reidemeister-Schreier algorithms are included. A recent addition to the supply of programs available constructs the

finite simple groups of order $<10^6$ both as permutation groups and by generators and relations.

LIBRARY PROGRAMS

As soon as one begins to develop programs in Cayley it beccomes imperative to store and edit programs and subroutines (or procedures as Cayley calls them). This is achieved through the use of a library file which has the form

LIBRARY name;
statements
FINISH;

Before entering the Cayley environment (for which the command is CAY) the programmer creates a library (or recalls one created at a previous working session). For example

PLIB groups/c

sets up such a library and calls it *groups*. Creating a library file called *permgp* and inserting it in *groups* requires

ADD permgp

The writer then edits the file in the usual way using the editor on the host machine. On exit from the editor Cayley conveniently prompts

Add problem to library (Y/N)?

(It calls library files problem files.) Modification can be achieved by

MOD permgp

which extracts the file from the library and allows it to be edited. In order to run *permgp* the user enters Cayley and then

LIBRARY permgp;

will execute the statements between the initial and final lines of the file. To change the file again it is necessary to exit from Cayley (using *QUIT;*) and then *MOD* again.

AN EXAMPLE

Given a finite group G and a subgroup H determine whether or not H is subnormal in G and if it is find its defect. We count the sequence of subgroups

$$G_0 = G,\ G_1 = H^{G_0},\ G_2 = H^{G_1},\ \ldots$$

and determine whether this series terminates in H or stabilizes at some larger subgroup. We avoid using the keyword *normal* *closure* in order to illustrate better the technique of nesting. The expression *invariant (U,V)* takes the Boolean value *true* if *U* is a normal subgroup of *V* and *false* otherwise. The complete program is placed in a library file called *defect*. Text entered within double quotation marks is regarded as comments and ignored by Cayley. A detailed listing of the program is given in the appendix.

HEANET

Finally we look briefly at the network aspect of using Cayley at UCG. At the time of writing HEANET connects UCG, UCC, UCD, TCD, NIHE (Dublin) and NIHE (Limerick). Local advice and permission is obviously required in order to avail of the network and in addition the prospective user will require permission and assistance from the Computer Centre at UCG (in particular to obtain a special command file for running Cayley). In practice the user calls UCG via his own computer terminal, logs on there as if his terminal were connected directly, carries out his working session and then reverses the procedure to break the connection. In the interests of economy it is important to minimise the time spent working with the network connected. This can be achieved by writing all

library files as text files in the local machine and using the *TRANSFER* option on the network to send them fully written to UCG. Illustrating with the example of the previous section the user would then have a file *defect.txt* in his directory at UCG. The command

LIBRARY/TEXT/LOG GROUPS defect.txt

(which is incidentaLLY part of VAX/VMS not Cayley) will then insert the file as a library file in the library *groups*.

CONCLUDING REMARKS

There is no doubt that this brief summary does not provide sufficient information for the reader to exploit fully the power of Cayley. However I hope that it will serve as an introduction to the rudiments. I will be happy to correspond in more detail with anyone who is interested and Ted Hurley at UCG has assured me that he is willing to help out with enquiries there.

REFERENCES

1. CANNON, John J.
   CAYLEY: A Language for Group Theory (Preprint 1982),
   University of Sydney.

2. CANNON, John J.
   "An Introduction to the Group Theory Language CAYLEY", in
   'Computational Group Theory', (Durham 1982), 145-183,
   Academic Press, L ndon - New York, 1984.

Mathematics Department,
University College,
Cork

APPENDIX

```
library defect;
"determine whether a subgroup H is subnormal in a group G and if it is
 find its defect"
print 'the group should be called G and the subgroup H';
print G,H;
U=G;
defect=0;
notdone = true;
"notdone is true until the seqeunce
       G0 = G, G1 = H^G, G2 = H^G1, ...
stabilises"
while notdone do
       V=H;
       notnorm = true;
"notnorm is true until the subgroup V is normal in U"
       while notnorm do
              for each x in generators(V) do
                 for each y in generators(U) do
                        V=<V,x^y>;
                 end;
                 if invariant(U,V) then
                        notnorm = false;
                 end;
              end;
       end;
"if V=U then sequence has stopped at a subgroup containing H strictly,
 if V=H then sequence has reached H,
 otherwise continue"
       if V eq U then
              break;
       else
              if V eq H then
                     notdone = false;
              end;
       end;
       defect = defect+1;
       U=V;
end;
"if notdone=true then the loop was broken, otherwise H is (sub)normal"
if notdone then
       print 'the subgroup is not subnormal';
else
       if defect eq 1 then
              print 'the subgroup is normal';
       else
              print 'the subgroup is subnormal with defect',
              defect;
       end;
end;
finish;
```