The Basis of a Computer System for Modern Algebra

John J. Cannon
Department of Pure Mathematics
University of Sydney

§1. Introduction

So-called general purpose systems for algebraic computation such as ALTRAN, MACSYMA, SAC, SCRATCHPAD and REDUCE are almost exclusively concerned with what is usually known as "classical algebra", that is, rings of real or complex polynomials and rings of real or complex functions. These systems have been designed to compute with elements in a fixed algebraic structure (usually the ring of real functions). Typical of the facilities provided are: the arithmetic operations of the ring, the calculation of polynomial gcd's, the location of the zeros of a polynomial; and some operations from calculus: differentiation, integration, the calculation of limits, and the analytic solution of certain classes of differential equations. For brevity, we shall refer to these systems as CA systems.

Up until about 1900, the various algebraic objects that had been studied were based on the real or complex numbers. However, the work of such people as Cayley in group theory, Weber in fields, Wedderburn in linear associative algebras, and Noether in ring and ideal theory, established the notion of abstract groups, fields, division algebras, rings and ideals. This type of algebra has come to be known as modern algebra. With this in mind I define a modern algebra system (or MA system) to be an algebra system which enables a user to compute in a wide range of different structures. Within reasonable bounds, an MA system should permit a user to define the structure in which he wishes to compute. To illustrate the distinction that I am attempting to draw between CA systems and MA systems, consider the problem of solving the system of linear equations

$$\Sigma_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1,\ldots,m .$$

where the a_{ij} and b_j are elements of a field K. In a CA system the user has to be satisfied with the one or two choices for K provided by the system (probably the rational field). In a true MA system the user would have a great deal of choice in specifying K. Thus, for example, he could take K to be a Galois field or a finite algebraic extension of the rationals.

A further distinction that can be made between CA systems and MA systems is that, whereas in CA systems the interest is in performing calculations with individual elements, in MA systems it is desirable to have the capability to compute some global properties of an algebraic structure. Here, I have in mind such things as the centre of a group, the radical of a ring, or a system of fundamental units for an algebraic number field.

To summarise then, a CA system works at the element level (locally) of an enormously rich but fixed algebraic structure, while an MA system allows the user to define the structure in which he wishes to work (subject to limitations imposed

by the implementation). Further, an MA system will often provide tools which can be used to investigate a structure globally.

§2. Issues in the Design of MA Systems.

While there is overlap in the problems faced by the designers of the two types of algebraic system, the designer of an MA system is faced with some novel problems. Chief among these must be the types of algebraic object (group, ring, field etc) permitted under the system, and, within a particular type of algebraic structure, the possible modes of specification (permutation group, matrix group etc). It is in this area that efficiency/generality trade-offs must be made. While it is possible, in principle, to provide the user with machinery sufficiently powerful to enable him to define and calculate in almost any algebraic structure, this would be achieved at the cost of unacceptable inefficiency. An alternative approach is to look at the ways in which algebraists describe structures of a particular type, and, from among these, choose the ones that appear to have the greatest versatility.

It is also necessary for an MA system to permit the coexistence of several distinct structures. For example, in group theory it is a standard technique when investigating a group G which is too complex to be investigated directly, to examine various homomorphic images onto less complex groups. Structure-preserving mappings play such an important role in abstract algebra that an MA system without some abilities to work with mappings would be seriously deficient.

I summarise below the more important issues in the design of an MA system:

(i) What kinds of algebraic structure will be included?

(ii) How will these various types of structure be represented?

(iii) How should simultaneous computation in distinct structures be organized?

(iv) How are sets of elements of these structures to be represented?

(v) What kind of mappings are to be permitted and how are they to be represented?

It should be noted that there are two parts to each of the above representation problems: What is the appropriate syntax to describe the object (in the user language of the system); and how is the object to be represented internally in the computer?

§3. The MA System Cayley

Over the past eight years I have been engaged in the development of what I believe to be the first serious attempt to build an MA system (Cannon [1,2,3]). This system is a package primarily intended for computing with discrete groups. Although it contains some facilities for working with infinite groups, the system is heavily biased towards finite groups. The user can specify a group either as a permutation group, as a group of matrices (over a finite field or the integers), or by a finite presentation. Apart from providing for element calculations, the system enables the user to compute such global information as conjugacy classes, normal subgroups, derived series, Sylow subgroups, centralizers and normalizers.

Since the study of matrix groups naturally involves the study of their actions on vector spaces and modules, Cayley allows the user to define and compute in various types of vector space and module.

As a system, Cayley comprises a language (see Cannon [2] for an early draft of the language), an interpreter, and a large library containing implementations of group theoretic and other algebraic algorithms. Apart from code written in Sydney, the library contains implementations of group theory algorithms produced by Neubüser's team in Aachen and Newman's team in Canberra.

Although Cayley was conceived as a group theory package, the fact that it allows computation in certain types of ring, field, vector space and module as well as in groups, means that the lessons learnt with Cayley have general applicability to the design of MA systems for areas other than group theory. In the remainder of this paper we discuss the principle features of the Cayley design.

§4. The Specification of Algebraic Structures

4.1. Groups

A basic decision taken early in the Cayley development was that the system should be powerful enough to compute with groups sufficiently large and complex to be relevant to contemporary research in finite group theory. For example, the classification of finite simple groups has thrown up computational problems involving permutation groups having degrees exceeding 1000, and, occasionally, very much larger degrees (Sims [5,6]).

In the most general sense a group is specified by defining a set, and a rule of composition for the elements of the set. The implementation of this general concept presents serious difficulties. Further, element multiplication would be extremely slow. For these reasons it was decided to restrict groups to those types which are widely employed and for which reasonable computation techniques are known. At present Cayley allows groups to be specified in the following ways:

(i) As a subgroup of the symmetric group Σ_n, for some integer n. In practice, n is restricted to around 10,000.

(ii) As a subgroup of the general linear group GL(n,K), where n is the dimension and K is an integral domain. Currently, Cayley allows K to be a finite field or the ring of integers. It is not difficult to extend the possibilities for K.

(iii) As a finite presentation by generators and relations.

4.2 Syntax for Groups

The definition of a permutation or matrix group is done in two distinct steps. The first step defines the *type* of permutation or matrix group with which we wish to work. The second step defines the actual permutation or matrix group by listing a set of generators for the group. It should be noted that for certain problems, the first step alone suffices. For example, it is sufficient to specify the group type if it is merely wished to form some products of particular elements.

The following two group specifications will serve to illustrate the flavour of the Cayley syntax.

```
(i)  G : PERM(8);
     G.GENERATORS: A = (1,2,3,4,6,7,8),
                   B = (2,3,5,7,8,6,4);
```

```
(ii) G : MATRIX(3,GF(2));
     G.GENERATORS: X = (1,0,0, 1,1,0, 0,1,1),
                   Y = (1,1,0, 0,1,1, 0,1,0);
```

Both examples define G to be the simple group PSL(2,7): the first defines it as the subgroup of Σ_8 generated by the permutations (1,2,3,4,6,7,8) and (2,3,5,7,8,6,4), while the second example defines it as the subgroup of GL(3,K), where K is the field of two elements, generated by the pair of matrices

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

As with permutation and matrix groups, the specification of a finitely presented group involves two steps. The first step lists the r generator symbols and the second gives the actual defining relations. In effect the first step defines the free group of rank r. We give the Cayley statements needed to define PSL(2,7) as the finitely presented group

$$\{R,S \mid R^3 = S^3 = (RS)^4 = (R^{-1}S)^4 = E\}$$

```
G : FREE(R,S);
G.RELATIONS : R↑3 = S↑3 = (R*S)↑4 = (R↑-1*S)↑4 = 1;
```

4.3 Fields

I shall confine myself to finite algebraic extensions of a field K. Let K(α): K be a simple algebraic extension of the field K, where α has minimum polynomial m(x) over K. Then any element of K(α) has a unique representation in the form p(α), where p is a polynomial over K and the degree of p(x) is less than the degree of m(x).

Further, the prime subfield of an arbitrary field K is isomorphic either to the field Q of rationals or the field Z_p of integers modulo a prime number p. Thus, calculation in finite algebraic extensions of either of these fields reduces to calculating with polynomials. Here it is desirable to have access to the polynomial facilities provided in most CA systems.

Finite fields present few difficulties. If the required finite field is small (i.e. having at most a few thousand elements) Cayley computes the table of Zech logarithms for the field.

At the time of writing no machinery has been provided for working with extensions of Q in Cayley. However, in the near future it is planned

to implement simple extensions of the rationals. I am loath to implement the general case in view of the fact that field calculations would often be extremely expensive.

The extension field K(a) of K is defined in Cayley using a statement of the form

```
L : FIELD(K,a,x,m(x))
```

where m(x) is the minimum polynomial of a over K. (It is assumed that the field K has been defined previously.)

4.4 Vector Spaces

The definition and representation of vector spaces over those fields definable in Cayley present no particular problems. Since all vector spaces of dimension n over a field K are isomorphic, it is convenient to take the space V(n,K) of n dimensional row vectors over K as the standard vector space. The type of vector space can then be specified by stating n and K. If desired, a basis can be supplied either for V(n,K) or for a subspace of V(n,K).

The following Cayley statements define the subspace of V(3,GF(9)) spanned by the vectors $(1,w,w^2)$ and (1,0,w), (where w is a primitive element of the field):

```
K : FIELD(Z(3),w,x,x↑3 + 2*x↑2 + 1);
V : VECTOR SPACE(3,K);
V.BASIS : X = (1,w,w↑2), Y = (1,0,w);
```

Vector spaces are an example of a structure which can be acted on by other structures. Thus, in Cayley, subgroups of GL(n,K) are permitted to act on V(n,K).

§5. Representation of Algebraic Structures

In the previous section the various types of algebraic structures that are specifiable in Cayley were outlined. In most cases the machine representation of their elements should be clear to the reader. How are the structures themselves to be represented in a computer? It might seem that having laid down rules as to the kind of structures permitted (e.g. permutation groups), the problem has already been settled. The problem, however, is more complex than this. I shall discuss the problem in terms of permutation groups. For certain computations with a permutation group, such as evaluating words in the generators, or testing whether the group is abelian, knowing the generators alone suffices. In order to answer deeper questions about the group, such as, for example, determining its conjugacy classes of elements, one needs to have available the set of its elements in some form. At this stage one might simply conclude that it is necessary to create and store the complete list of elements of the group. However, if one analyzes how the set is actually used, it becomes clear that a large amount of group theoretic computation can be carried out provided that the following operations are available:

(i) Determine the order of G;

(ii) Determine if an element of the symmetric group Σ_n is in G;

(iii) Run through the elements of G without repetition.

In [4], Sims introduced the notion of *base* and *strong generating set* and used them to give a very elegant solution to this problem for permutation groups.

The great majority of permutation group algorithms have been designed to utilize the strong generating set concept. Two points should be made here: firstly, the cost of obtaining a strong generating set is often high compared to the cost of subsequent computation (e.g. computing a centralizer); and secondly, in a typical session computing with a permutation group, one tends to deploy a series of algorithms that assume the availability of a strong generating set for the group. It is therefore clear that once a strong generating set has been calculated for G, then it should be retained throughout the computation.

In Cayley I have adopted the philosophy of saving not only such critical information as strong generators, but also other information that may be useful in subsequent computation. Thus, for example, if the classes are computed for any reason they will be automatically saved. Any subsequent calculation requiring the conjugacy classes will check to see if they are already known before attempting to construct them.

It is necessary then to keep track of the information that has accumulated for an algebraic

structure. This is done by means of a *structure table*, one of which is associated with each algebraic structure stored in the machine. This table stores such information as:

(i) Type of structure;

(ii) Definition of structure;

(iii) Method of representing individual elements;

(iv) Representation of the set of elements (e.g. strong generating set plus associated information in the case of a permutation group).

(v) The properties and structural information already computed which could be useful in subsequent calculations.

A Cayley computation, then, can be viewed, from the point of view of the machine, as the construction of a rather complex data structure. As new facts are discovered about the object, they are added to this data structure. As the computation progresses it is expected that at least some of the users' requests can be met by doing little more than extracting information from this data structure. The construction of this data structure is invisible to the user.

It is necessary to distinguish between information that clearly belongs to a single structure (e.g. classes of a group) from information whose definition involves two or more structures (e.g. coset table of a subgroup). In the course of computing with an algebraic structure A, it is usual to generate substructures of A and homomorphic images of A. The relationship between A and a substructure or homomorphic image of A must be noted, and there must be provision for the storage of information common to a pair of structures related in such a manner. In Cayley this is achieved by means of a *structure relationship table*.

Finally, given a request from a user, the system must have suitable machinery which matches the desired information with what is currently known and then produces a "micro program" to compute the answer as economically as possible.

I conclude this section by summarizing the issues raised:

(i) Very compact methods for representing the set of elements of certain types of group are known.

(ii) Having computed certain information about an algebraic structure, it is often desirable to retain it so as to avoid subsequent recalculation.

(iii) An algebraic structure is stored in the machine by means of a structure table. This table stores not only the definition of the structure but also what is known about the structure and its whereabouts.

(iv) Information common to a pair of related structures is stored in a structure relationship table.

(v) Sophisticated machinery is necessary to properly utilize the data structure which is constructed during a computation session.

Acknowledgement

This work was supported by a grant from the Australian Research Grants Committee.

References

1. J.J. Cannon: A general purpose group theory program. Procs. 2nd Internal. Conf. Theory of Groups. Lecture Notes in Math., Vol. 372, Springer, Berlin, 1974, 204-217.

2. J.J. Cannon: A draft description of the group theory language Cayley. SYMSAC'76. Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation. Association for Computing Machinery, New York, 1976, 66-84.

3. J.J. Cannon: Software tools for group theory. Proceedings of a Symposium in Pure Mathematics, 37, American Mathematical Society, Providence, R.I. 1981.

4. C.C. Sims: Computational methods in the study of permutation groups. Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967). Edited by J. Leech, Pergamon, Oxford, 1970, 169-183.

5. C.C. Sims: Some group-theoretic algorithms. Topics in Algebra. Lecture Notes in Math., Vol. 697, Springer, Berlin, 1978, 108-124.

6. C.C. Sims: A method for constructing a group from a subgroup. Topics in Algebra. Lecture Notes in Math., Vol. 697, Springer, Berlin, 1978, 125-136.