A Deductive Database of the Groups of Order Dividing 128

Greg Butler, Sridhar S. Iyer, and Susan H. Ley
Basser Department of Computer Science
University of Sydney
Sydney NSW 2006 Australia

Abstract

This paper describes the design and implementation of a deductive database for the 2668 groups of order 2^n, $(n \leq 7)$. The system was implemented in NU-Prolog, a Prolog system with built-in functions for creating and using deductive databases. In addition to the database, a simple query language was written. This enables database users to access the data using a simpler and more familiar set-theoretic syntax than that provided by the Prolog interpreter.

Introduction

The work described here is a feasibility study of the application of deductive database technology to mathematical information for eventual integration with a computer algebra system. To the best of our knowledge, it is the first deductive database for group theory.

Cayley, version 4, designed by Butler and Cannon(1989), is a proposed knowledge-based system for modern algebra. The proposal integrates the existing powerful algorithm base of Cayley (see Cannon, 1984) with modest deductive facilities and large sophisticated databases of groups and related algebraic structures. The motivation for these changes includes the need to provide facilities for a knowledge-based system, and to allow sets to be defined by properties. The database queries in Cayley are modelled on the set constructors which allow a set to be defined by properties or conditions.

For example, a mathematician might ask

> *Determine all the groups of order 128 which have a centre of order 4, a nilpotency class of 3, and precisely 12 elements of order 8.*

In Cayley, version 4, if the user language variable *twogps* referred to the database then the query would be

```
s := { G in twogps | order(G) eq 128
and order(centre(G)) eq 4
and nilpotency class(G) eq 3
and #{ x in G | order(x) eq 8 } eq 12 };
```

In our Prolog deductive database, the query would be

```
s := { G in twogps | order(G) eq 128
and order(centre(G)) eq 4
and nilpotency class(G) eq 3
#elements order(G,8) eq 12 };
```

Furthermore, the query could be generalized to determine those groups of order 128 where the number of elements of order 8 is the product of the nilpotency class and the order of the centre.

```
s := { G in twogps | order(G) eq 128 and
#elements order(G,8) eq (order(centre(G)) *
    nilpotency class(G)) };
```

The rationale for such a knowledge-based system comes from three sources. Firstly, there is the growing number of libraries or catalogues of groups classified by various properties. A common task of an algebraist is to seek examples and counterexamples to conjectures from amongst these catalogues. Therefore, database facilities are required. Secondly, there has always been the desire to fully utilise the user's knowledge of a problem. To date only the algorithmic knowledge has been applied in the solution of a problem. Thirdly, the internal planning of computations inside Cayley in order to guarantee that prerequisite data is available for each step of a calculation has become increasingly complex. It is felt that a rule-based planner running on an inference engine would help control the complexity of the task. This suggested we consider deductive databases.

The overall project is investigating the technology of Prolog deductive databases as an appropriate method of providing the database and inference engine facilities in Cayley. There are (at least) two dimensions which we

need to investigate. One is the complexity and variety of algebraic objects, and the other is the sheer volume of data that is available. The first is investigated by Butler and Iyer(1990), who are constructing a database and knowledge base of the 56 simple groups of order less than one million. The project described here considers the second dimension in that it is the prototype for a database of the 58,760 groups of order dividing 256. The final database will have well over 1,000,000 pieces of information.

We chose a set of data that already existed as a Cayley library (Newman and O'Brien, 1989) and a Pascal file management system (James, Newman, O'Brien, 1990). The collection is large, but homogeneous in that the information can readily be encoded as (small) integers.

The major aims of this database project are

- To examine the appropriateness of Prolog to represent the data.
- To examine the appropriateness of Prolog as a query language.
- To provide a convenient interface between the user and the database system.
- To examine the feasibility of translating the Cayley set-theoretic database query language into Prolog queries.
- To examine the performance of the Prolog database.

The project is implemented in NU-Prolog (Thom and Zobels, 1987; Thom and Naish, 1983) because of its deductive database facilities, its availability, and our contact with its authors at the University of Melbourne. It has a built-in database system and a definite clause grammar processor that can be used for implementing query languages.

Background on Computer Algebra

Computer algebra and computer algebra systems (see Buchberger, Collins, Loos, 1982) have been primarily concerned with algorithms and data structures for algebraic objects such as polynomials, power series, groups, and fields. Gradually, the importance of the user language, particularly its expressibility and extensibility, has become evident. Modern languages such as Scratchpad of Jenks(1984), and the language proposed by Calmet and Lugiez(1987) have strong typing, generic abstract data types (or categories) with inheritance and enrichment, and type inference. However, little attention has been paid to the information storage and retrieval aspects of the problem domain.

Modern algebra has a long history of classification and cataloguing of examples. There is a growing number of lists or catalogues of groups classified by various properties. For example,

- the Hall and Senior(1964) catalogue of the 2-groups of order up to 64,
- the list of primitive permutation groups of small degree by Sims(1970),
- the character tables of the simple groups of order less than one million by McKay(1979),
- the Atlas of Conway, Curtis, Norton, Parker and Wilson(1985) of the sporadic simple groups,
- the list of groups of order 128 compiled by James, Newman, and O'Brien(1990),
- the list of groups of order 256 compiled by O'Brien(1988), and
- the perfect groups of order up to one million (Holt and Plesken, 1989).

Furthermore, many of these catalogues are available online through file management systems of limited sophistication. For example,

- Cayley has had libraries of examples (accessible by file name) since 1975;
- the character tables of the Atlas are on the computer at Cambridge (UK); and
- the CAS system of Neubüser, Pahlings, and Plesken(1984) has files containing the character tables of an extensive collection of groups.

The literature also contains parametric descriptions of (some) properties of groups, such as the Lie groups, which belong to infinite families. Typical parameters are d and q where the underlying vector space is a d-dimensional vector space over GF(q). The properties include matrix or permutation generators, presentations, and character tables. Cayley has built-in functions which return instances of the groups and their definitions by presentations or generators. Similarly CAS can create an instance of a character table from the known parametric descriptions.

While some projects (see Worboys,1988) have the aim of creating databases of groups, they have only created file management systems. Their query languages are much too limited to be considered databases.

Background on Prolog and Deductive Databases

Deductive databases are databases with the ability to make deductions from the facts and rules they contain. They usually consist of large amounts of data and a small number of rules which allow other facts to be deduced. Lloyd(1983) provides an introduction to deductive databases in Prolog, and some of their features. The major differences between an ordinary Prolog program, containing both facts and rules, and a deductive database is a practical one. A deductive database contains large number of facts, usually too many to be stored in main memory, and only a relatively small number of rules. Lloyd considers the problems of query optimization and clause indexing that this causes. The clause indexing problem is reduced to one of an efficient partial-match retrieval scheme.

Thom and Naish(1983) describe a deductive database system implemented using MU-Prolog, a predecessor of NU-Prolog which was used for this project. NU-Prolog utilizes superimposed code words for clause indexing. The idea behind superimposed coding is that each record (of the database) is described by a code word. Each field of the record generates a string of bits set to either one or zero depending on the value of the field. After a code word has been generated for each field they are or-ed (or superimposed) to form the code word of the record. Ramamohanarao and Shepherd(1986) give a detailed description of a form of superimposed coding which is suitable for dealing with large databases of Prolog clauses. This is the method implemented in NU-Prolog.

Definite clause grammars (Pereira and Warren, 1980) are generalizations of context free grammars and are suitable for natural language processing. They provide context-dependency; allow arbitrary tree structures to be built up during the course of parsing; and allow conditions to be included in the grammar rules.

System Overview

The database may be viewed as a database with one of two query languages. The first is just the language of Prolog which directly accesses the relations in the database. The second is a set-theoretic query language modelled on the set constructors of Cayley. A help facility is provided to inform users about the Prolog relations -their names and attributes- and about the functions and operators of the set-theoretic language. As most users will access the database only occasionally, such a help facility is important.

The relations are stored on disc and retrieved using superimposed coding to solve the partial match retrieval problem. Our database is not dynamic so we use static superimposed coding (SIMC).

The set-theoretic query language is parsed into a Prolog goal. The grammar of the language is specified by a definite clause grammar (DCG) with actions to generate the appropriate Prolog goal. Hence, the parser is a Prolog program, which is the compiled form of the DCG.

The overall architecture of our database is shown in Figure 1.

The Data

Each group G in the database is denoted by an identifier *Order#Number*, which indicates that G is the *Number*-th group of order *Order* in the list compiled by James, Newman, and O'Brien. A group of order 2^n may be defined by a power-commutator presentation (pcp) of the form

$$\begin{aligned}
\langle a_1, a_2, \cdots, a_n \mid {a_i}^2 &= \prod_{k=i+1}^{n} {a_k}^{\epsilon(i,i,k)}, \qquad 1 \le i < n, \\
{a_n}^2 &= \mathit{identity}, \\
(a_j, a_i) &= \prod_{k=j+1}^{n} {a_k}^{\epsilon(i,j,k)}, \quad 1 \le i < j < n, \\
(a_n, a_i) &= \mathit{identity}, \qquad 1 \le i < n \rangle
\end{aligned}$$

The exponents $\epsilon(i,j,k)$ are either 0 or 1. There are $(n+1)(n-1)n/6$ exponents. The sequence of exponents $\epsilon(i,j,k)$, $1 \le i \le j < k \le n$ may be regarded as the binary representation of an integer. For $n = 7$, the integer may be as large as $2^{56} - 1$. In order to constrain the integer data in the database to be within the limits of the Prolog system, the integer is decomposed into 4 digits taken base 2^{14}. These 4 small integers are a compact encoding of the pcp. For similar reasons, we must decompose the order of the automorphism group $Aut(G)$ of G into small integers.

There are four relations in the database:

- group_info/15,
- seq/9,
- num_of_classes/4, and
- elements_of_order/4.

Each of the 2668 groups is represented by a set of clauses. In total there are 20217 num_of_classes facts, and 18228 elements_of_order facts. We describe each relation below.

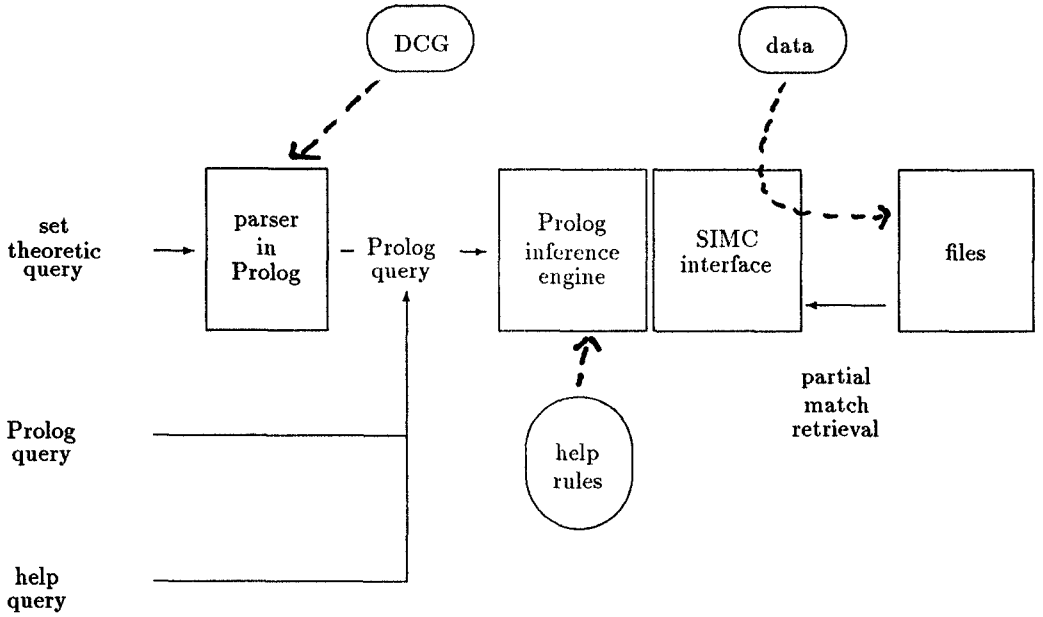

Figure 1: System Architecture

The relation

group_info(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o)

was designed to hold information that is present for all groups, and is present only once for each group. The arguments are

a#b is the group
c#d is the parent group
e#f is the central quotient
g#h is the centre
i#j is the commutator subgroup
k is the number of defining generators
l is the nilpotency class
m is the order of $Aut(a\#b)$ div 10^7
n is the order of $Aut(a\#b)$ mod 10^7
o is the isoclinism family number

The relation

seq(a, b, c, d, e, f, g, h, i)

contains exactly the same information as the sequences in the Cayley library of 2-groups (see Newman and O'Brien, 1989). The arguments are

a#b is the group
c is the number of defining generators
d is $log_2(a)$, the number of pcp generators
e is the exponent-p class
f,g,h,i are integers encoding the pcp

The relation

num_of_classes(a, b, c, d)

contains the information about the number of classes of a given length in a given group. The arguments are

a#b is the group
c is the class length
d is the number of conjugacy classes of
length c in the group

The relation

elements_of_order(a, b, c, d)

contains the information about the number of elements of a given order in a given group. The arguments are

a#b is the group
c is the order
d is the number of elements of
order c in the group

Here as an example is the data stored about 32#26, the 26-th group of order 32.

```
group_info(32,26,8,5,4,2,8,2,2,1,3,2,0,384,2)
      seq( 32, 26, 3, 5, 2, 4097, 5632, 0, 0 )
      num_of_classes(32, 26, 1, 8)
      num_of_classes(32, 26, 2, 12)
      num_of_classes(32, 26, 4, 0)
      num_of_classes(32, 26, 8, 0)
      elements_of_order(32, 26, 2, 3)
      elements_of_order(32, 26, 4, 28)
      elements_of_order(32, 26, 8, 0)
      elements_of_order(32, 26, 16, 0)
      elements_of_order(32, 26, 32, 0)
```

There are several rules which extract attributes from the raw facts. These are illustrated in Figure 2. Some are simply views of the data, such as

```
central_quotient_of4(Ord,Num,Cord,Cnum) :-
   group_info(Ord,Num,_,_,Cord,Cnum,
              _,_,_,_,_,_,_,_,_,_).

central_quotient_of(Ord,Num) :-
  group_info(Ord,Num,_,_,Cord,Cnum,
             _,_,_,_,_,_,_,_,_,_),
  write('The central quotient is : '),
  write(Cord), write(' # '), writeln(Cnum).
```

some are display rules, and *tot_classes* is an aggregation rule summing the fourth attribute of *num_of_elements*.

```
tot_classes(Ord,Num,Ans) :-
      tot_classes1(Ord,Num,1,Ans).

tot_classes1(Ord,Num,N,Ans) :-
      num_of_classes(Ord,Num,N,A),
      Ans is A + A1,
      N1 is N * 2,
      tot_classes1(Ord,Num,N1,A1).

tot_classes1(Ord,Num,Ord,0).

tot_classes1(Ord,Num,N,Ans) :-
      N1 is N * 2,
      tot_classes1(Ord,Num,N1,Ans).
```

We could also have defined rules to deduce other properties, such as being abelian or elementary abelian, as follows:

```
abelian(Ord, Num) :-
      num_of_classes(Ord,Num,1,Ord).

elementary_abelian(Ord, Num) :-
      N is Ord - 1,
      elements_of_order(Ord,Num,2,N).
```

It should be noted that there are some redundancies in the data stored for a group. The number of conjugacy classes of length 1 is equal to the order of the centre, so the fact

```
num_of_classes(32, 26, 1, 8)
```

for 32#26 is redundant. If a group has no elements of order 2^m then it cannot have any elements of order 2^{m+1} so the facts

```
elements_of_order(32, 26, 16, 0)
elements_of_order(32, 26, 32, 0)
```

for 32#26 are redundant. Indeed, we could have stored only the facts *elements_of_order(a,b,c,d)* where *d* is non-zero, and implied the zero value of the attribute from the failure of the Prolog goal. We chose to retain these redundancies to simplify the query processing, and to support future plans to include them in integrity constraints on the database. On the other hand, it is well-known that only the identity element of a group has order 1, and queries requesting this data would seem most unlikely, so this fact is implicit in the database.

The Help Facility

For novice or irregular users of a system a help facility is important. It indicates the range of information available, the Prolog predicate definitions, and the meaning of each of the many attributes within the predicates. The help facility is provided by two predicates:

- help/0, which provides an overview of the database (see Figure 2), and
- help_on/1, which provides details of a predicate (see Figure 3 for an example).

Both display appropriate textual information.

```
THE TWO-GROUPS DATABASE
-----------------------
This database contains information on groups of
order 2^n (n <= 7). The data can be accessed in
various ways. The help command provides a list
of the available predicates.
The following predicates are available -
    group/2
    all_about/2
    exponent_pclass/2
    exponent_pclass3/3
    commutator_subgp_of/2
    commutator_subgp_of4/4
    centre_of/2
    centre_of4/4
    central_quotient_of/2
    central_quotient_of4/4
    parent_of/2
    parent_of4/4
    tot_classes/3
    seq/9
    elements_of_order/4
    num_of_classes/4
    group_info/15
For information on one of these predicates use
    help_on(predicate name).
```

Figure 2: The Help Screen

The Query Language

The users of Cayley and the database are mathematicians. They are very familiar with the usual notations of set theory and algebra such as

$$\{ x^2 : x \in S \mid x \text{ is even } \}$$

which determines the squares of all even integers in S.

The user language of Cayley, version 4, has similar set constructors,

$$\{ x \hat{} 2 : x \textbf{ in } S \mid \text{even}(x) \}$$

and, rather than use a separate database query sublanguage, the queries are based on set constructors.

The query language for the deductive database is a slightly simplified form of that for Cayley, version 4. The query language provides statements to

1. name the database;
2. query the database; and
3. print solution sets.

An identifier may be associated with a database using the statement

```
2?- help_on(all_about).
Usage
-----

all_about(<order>, <number>)

Description
-----------

Prints out all the information in the database
about the group. The information is printed in
the format used by the Pascal database
written by O'Brien.
```

Figure 3: An Example of help_on

<db_id> := **database**(<Prolog_database_name>);

Then the database may be queried using

<set_id> := { <id> **in** <db_id> | <condition> };

to return a set of solutions of all objects that are in the database specified and also satisfy the condition.

This solution set may be printed using the print statement

print <set_id>;

The query language is described by a definite clause grammar. The associated parser translates the query into a Prolog goal, and performs some minor optimizations at the same time. The query language parser is entered by calling the predicate *query/0.*

```
1?- query.
Query> d := database( twogps );
Query> s := { g in d | centre(g) eq 4#1 and
central quotient(g) eq 16#11 };
Query> print s;
      [(64,102), (64,111), (64,125), (64,256)]
```

Figure 4: Examples of the Query Language

The condition in a query consists of boolean expressions. These expressions may call the functions listed in Figure 5 as well as permissible operators. The functions have an argument X, which is either the identifier <id> from the query which symbolizes a general group in the database, or a specific group designated by its identifier *Order#Number.* The expressions follow the usual (default) precedence of operators, and bracketting may be used to ensure the correct precedence.

```
The functions available are
   #classes(X)
   #classes length(X, Length)
   #elements order(X, Order)
   isoclinism(X)
   automorphism group order(X)
   central quotient(X)
   parent(X)
   centre(X)
   commutator subgroup(X)
   #defining generators(X)
   order(X)
   exponent pclass(X)
   nilpotency class(X)

Arithmetic expressions involving functions may
appear using the operators *, /, +, -

The available connectives are
   not, and, or        the logical operators
   eq, gt, ge, lt, le  the comparison operators
```

Figure 5: The Query Language Functions and Operators

As an example of the expressibility of the query language, the three queries in James, Newman, and O'Brien(1990) can be stated as

```
s1 := { G in twogps | order(G) eq 128 and
   #defining generators(G) eq 4
   and #elements order(G,8) eq 0
   and #elements order(G,4) gt 0
   and nilpotency class(G) eq 3
   and order(centre(G)) eq 2 };

s2 := { G in twogps | order(G) eq 128 and
   central quotient(G) eq 16#14 and
   commutator subgroup(G) eq 8#5
   and #classes length(G,8) eq 12
   and #classes(G) ge 20
   and #classes(G) le 32 };

s3 := { G in twogps | order(G) eq 128 and
   automorphism group order(G) eq 4096
   and #elements order(G,2) eq 7
   and #elements order(G,16) eq 32
   and nilpotency class(G) eq 4
   and order(centre(G)) eq 8 };
```

Note that in the first query we must paraphrase the condition that the exponent of G is 4, using the number of elements of order 4 and 8.

There are some unnecessary limitations on the query language. It is hoped to allow <set_id> as well as <db_id> in the query statement so one can refine a previous solution set. It is also hoped to allow function applications to be nested. At present, only the *order* function may have a general expression as an argument. The other functions must have simple variables or constants as arguments.

The translation process is slow - up to two minutes on a Sun 3/60 for a moderately complicated query such as

```
s := { G in twogps | order(G) eq 128 and
#elements order(G,8) eq (order(centre(G)) *
   nilpotency class(G)) };
```

The solutions to the Prolog goal generated by the translation process are found quickly if specific information (in the form of integer constants) is supplied. For example,

```
s := { G in twogps | centre(G) eq 4#1 and
     central quotient(G) eq 16#11 };
```

requires 9 seconds. However, for queries which specify relations between attributes, as in

```
s := { G in twogps | order(centre(G)) eq order(G)/4 };
```

the solutions may not be found quickly. This example requires about 4 minutes. In the latter case, all the groups are retrieved and then the relation is tested. We would like to improve this aspect by refining the goal produced by the parser. The database is static, and the range of values and their frequency can be determined. Knowing the range of values that an attribute can take would allow us to refine the previous query to

```
s := { G in twogps |
order(G) eq 8 and order( centre(G) ) eq 2
or order(G) eq 16 and order( centre(G) ) eq 4
or order(G) eq 32 and order( centre(G) ) eq 8
or order(G) eq 64 and order( centre(G) ) eq 16
or order(G) eq 128
       and order( centre(G) ) eq 32 };
```

which requires 4 seconds rather than 4 minutes to find the solutions. The information on frequency of attribute values would allow optimisation of queries through conjunct re-ordering.

Conclusion

The development of the deductive database has demonstrated

- that Prolog and superimposed coding are appropriate technologies to use for this particular database;
- that a familiar set-theoretic query language can be provided;
- that translation between the set-theoretic query language and Prolog is practical.

So, in particular, an integration of the deductive database facilities with Cayley is feasible.

The limitations of the current database, which we will address in future work, are

- that studies (and possible improvements) in performance are required. No experimentation with the choice of parameters for superimposed coding has been undertaken.
- that translation of the query language be improved. The functionality of the query language does not allow nesting of function calls (other than *order*). Only minimal attempts to optimize queries has occurred. The translation is slow, and a traditional parser written in C is one solution to this problem.
- that the 56,092 groups of order 256 are not in the database yet. This may impose further impetus to tune performance.

Acknowledgements: A number of people provided expertise on NU-Prolog and deductive databases - John Shepherd, Rodney Topor, Ron Sacks-Davis - while the data on 2-groups was generously made available by Eamonn O'Brien.

This work was supported in part by the Australian Research Council.

References

B. Buchberger, G.E. Collins, and R. Loos, **Computer Algebra : Symbolic and Algebraic Computation,** Springer-Verlag, Wien, 1982.

G. Butler and J.J. Cannon, *Cayley, version 4 : the user language,* **Symbolic and Algebraic Computation,** P. Gianni (ed.), *Lecture Notes in Computer Science* **358**, Springer-Verlag, Berlin, 1989, pp.456-466.

G. Butler and S.S. Iyer, *Deductive mathematical databases - a case study,* **Statistical and Scientific Database Management,** Z. Michalewicz (ed.), *Lecture Notes in Computer Science* **420**, Springer-Verlag, Berlin, 1990, pp. 50-64.

J. Calmet and D. Lugiez, *A knowledge-based system for computer algebra,* SIGSAM Bulletin, **21**, 1 (1987) 7-13.

John J. Cannon, *An introduction to the group theory language, Cayley,* **Computational Group Theory** M. D. Atkinson, editor, Academic Press, London, 1984, 145-183.

W.F. Clocksin and C.S. Mellish, **Programming in Prolog,** Springer-Verlag, Berlin, 1981.

J.H. Conway, R.T. Curtis, S.P. Norton, R.A. Parker, R.A. Wilson, **Atlas of Finite Groups,** Clarendon Press, Oxford, 1985.

M. Hall, Jr and J.K. Senior, **The Groups of Order 2^n, (n <= 6),** Macmillan, New York, 1964.

D.F. Holt and W. Plesken, **Perfect Groups,** Oxford University Press, 1989.

R. James, M.F. Newman, E.A. O'Brien, *The groups of order 128,* J. Algebra, **129**, 1 (1990) 136-158.

R.D. Jenks, *A primer : 11 keys to New SCRATCHPAD,* **EUROSAM 84,** J. Fitch (ed.), *Lecture Notes in Computer Science,* **174**, Springer-Verlag, 1984, 123-147.

S.H. Ley, **TWOGPS : A Deductive Database for Groups of Order 2^n.** Honours Thesis, University of Sydney, 1988.

J.W. Lloyd, *An introduction to deductive database systems,* Australian Computer Journal **15**, 2 (1983) 52-57.

J.K.S. McKay, *The non-abelian simple groups G, $|G| < 10^6$ - character tables,* Communications in Algebra, **7**, 13 (1979) 1407-1445.

J. Neubüser, H. Pahlings, and W. Plesken, *CAS; Design and use of a system for the handling of characters of finite groups,* **Computational Group Theory** M. D. Atkinson, editor, Academic Press, London, 1984, 195-247.

M.F. Newman and E.A. O'Brien, *A Cayley library for the groups of order dividing 128,* **Group Theory (Singapore, 1989),** Walter de Gruyter, Berlin, New York, 1989, pp. 337-342.

E.A. O'Brien, **The Groups of Order Dividing 256,** Ph.D. Thesis, Australian National University, 1988.

F.C. Periera and D.H.D. Warren, *Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks,* Artificial Intelligence **13** (1980) 231-278.

K. Ramamohanarao and J. Shepherd, *A superimposed codeword indexing scheme for very large Prolog databases,* Proceedings of the Third International Conference on Logic Programming, London, 1986, pp. 569-576.

C.C. Sims, *Computational methods for permutation groups,* **Computational Problems in Abstract Algebra,** (Proceedings of a conference in Oxford, 1967), J. Leech (ed.), Pergamon Press, Oxford, 1970, pp.169-183.

J.A. Thom and L. Naish, *The MU-Prolog deductive database,* TR 83/10, Department of Computer Science, University of Melbourne, 1983.

J.A. Thom and J. Zobels (eds), *NU-Prolog Reference Manual ver 1.1,* TR 86/10, Department of Computer Science, University of Melbourne, 1987.

M.F. Worboys, *Groups - a database for computational algebra,* **Computers in Mathematical Research,** N.M. Stephens and M.P. Thorne (eds), Clarendon Press, Oxford, 1988, pp.153-160.