LOGIC AND PROLOG DATABASE SYSTEM

Mohamed bin Othman
(Computer Science Unit) Mathematics Department
University Pertanian Malaysia
43400 UPM Serdang Selangor
Malaysia

Synopsis

This paper discusses an initial research of a subfield of first order predicate logic applied to the database. Consideration has been made toward the relational database system. Here, the logic used boths as an inference system as well as a representation language. The use of logic for knowledge representation and manipulation is previously due to the work of question-answering system, which have been mainly concerned with the deductive manipulation of a small set of facts and thus require an inferential mechanism provided by logic. Similar techniques have been adopted to databases to handle large set of facts, open queries, and others.

Keywords: Relational database, Programming in logic, classical interpretation, unification and queries.

Introduction

PROLOG or PROgramming in LOGic is a higher-level programming language developed by Colmerauer A., et al 1977 at Marseilles University, France. Curiously enough, Hungary was one of the earliest countries to recognise its potential more than pure academic research. Since that time its adoption as the programming language for Japan's Fifth Generation Project has established its reputation worldwide.

It is based on the first order predicate logic and has been used for applications of symbolic computation. It operates on tree structured data named, terms and composed of symbolics, variables and numbers. The basic operation is unification, which comparing two terms and trying to make equal by assigning values to the variables. If it is succesful, the result is then the list of subtitutions made on the variables.

A database is a collection of interelated data stored without redundancy to serve one or more applications in an optimal fashion. The data is stored so that it is independent of the programs which use it.

The architecture of a database management system as state in Date C. J., 1981, is divided into three levels;

1) The conceptual level (Logical database).
   These level consists of the abstract representation of the database (i.e. independent from the physical implementation),

2) The internal level (Physical database).
   It is the implementation of the logical database and concerned with there presentation of data type record formats, storage structures and access methods. It is there presentation of the database as is actually stored and retrieved,

3) The external level (External database).
   It is concerned with views of the logical database as seen by the users. Each view consists of some portion of the logical database.

There are three main approaches have been used to specify a conceptual model for database systems. These are hierarchical, network and relational approaches. Consideration has been made only on the relational approach, and it is because of their relational model of data that interrelated between the mathematical logic and data base are concerned. In the relational model, data are organised as a collection of relations. Gallaire H., et al 1977 state that a relation can be defined mathematically as follows;

> "Let $D_1, D_2, \ldots, D_n$ be n domains (not necessarily distinct) of elements. A relation defined on $D_1, D_2, \ldots, D_n$ is a subset of the cartesian product $D_1 \times D_2 \times \ldots \times D_n$"

In a relational database, each relation is represented by a table and each column of the tabular relation corresponds to a field (or attribute). Obviously, the order or the components in the tuples is meaningful. The relations of a database are best expressed in a *"normal form"* which eliminates redundancy and simplifies updating procedures. The various normal forms are discussed in standard text such as Date C. J., 1981.

Mathematical Logic

Mathematical logic has been applied to many different areas, including that of databases. It can be defined from two different viewpoints; the semantic view and syntatic view as state by Gallaire H. et al 1977. Both approaches are founded upon first defining a language as a collection of symbols and rules for building well-formed formulas(WFFs).

A particular form of logic is called the first order predicate calculus or predicate logic. These predicate logic consist of the following primitive symbols;

1) Individual symbols.
   In other words; it consists of a variable and a constant,
2) Function symbols.
   Which will be denoted by lower case letters such as f, g,..., etc,
3) Predicate symbols.
   Which will be denoted by upper case letters such as P, Q,..., etc,
4) Logical symbols.
   Such as $\wedge$ (and), $\vee$ (or) and $\neg$ (not),
5) Quantifier symbols.
   Such as $\forall$ (for all) and $\exists$ (that exist),
6) Punctuation symbols.
   Such as ",", "(" and ")",

A term is defined (recursively) to a constant or a variable, or if f is a n-arys function and t_1, t_2,... t_n are terms, then $f(t_1, t_2,..., t_n)$ is a term and there are no other terms.

If P is an n-arys predicate symbol and t_1, t_2, ..., t_n are terms, then $P(t_1, t_2,..., t_n)$ is an atmoic formula. When $n = 0$, the atomic formula is called a proposition. An atomic formula or its negation, will be refered to as a literal.

Well-formed formulas are defcined using atomic formulas, parentheses, logical connectives, universal quantifiers and existential quantifiers as follows;

1) Atomic formulas are well-formed formulas(WFFs),
2) If A is a WFF and y is an individual variable, then $(\exists y)A$ is a WFF,
3) If A and B WFFs and y is an individual variable, then the following are also WFFs.
   a) $\neg A$ Not (negation).
   b) $A \vee B$ Or (disjunction).
   c) $A \wedge B$ And (conjuction).
   d) $A \Rightarrow B$ Implication i.e. if A sufficinet for B.
   e) $A \Leftrightarrow B$ Equivalence i.e. if and only if A is sufficient and necessary for B.
   f) Universal quantifier, example $(\forall y)(A(Y) \Rightarrow B(y))$.
   g) Existential quantifier, example $(\exists y)(A(y) \Rightarrow B(y))$.

The above definitions defined those string of symbols which are valid statement in the first order predicate calculus and rule out the others. For instance,

$\forall x(\exists y(P(x, y) \quad Q(y) \Rightarrow R(x,y)))$ is a WFF.
$\forall)xQRS(\exists)Y$ is not a WFF.

One can define and, or and equivalence in terms of truth tables. It is also posibble to define implication such as:

$(A \Rightarrow B) \Leftrightarrow (\neg A \quad B)$.

Some of the well known equivalence are given below;

1) Simple equivalences.
   $\neg(\neg x) \Leftrightarrow x$
   $x \vee y \Leftrightarrow x \Rightarrow y$
2) De Morgan's Laws.
   $\neg(x \wedge y) \Leftrightarrow \neg x \vee \neg y$
   $\neg(x \vee y) \Leftrightarrow \neg x \wedge \neg y$
3) Distributive law.
   $x \wedge (y \vee z) \Leftrightarrow (x \wedge y) \vee (x \wedge z)$

$x \vee (y \wedge z) \Leftrightarrow (x \vee y) \wedge (x \vee z)$

4) Commutative laws.
   $x \wedge y \Leftrightarrow y \wedge x$
   $x \vee y \Leftrightarrow y \vee x$

Equivalence with qualifications.

1) $\neg(\exists x)P(x) \Leftrightarrow (\forall x)(\neg P(x))$
2) $\neg(\forall x)P(x) \Leftrightarrow (\exists x)(\neg P(x))$
3) $(\forall x)(P(x) \wedge Q(x)) \Leftrightarrow (\forall x)P(x) \wedge (\forall y)Q(y)$
4) $(\exists x)(P(x) \vee Q(x)) \Leftrightarrow (\exists x)P(x) \vee (\exists y)P(y)$
5) $(\forall x)P(x) \Leftrightarrow (\forall y)P(y)$
6) $(\exists x)P(x) \Leftrightarrow (\exists y)P(y)$

It is possible for any WFF in the first order predicate calculus to be converted into the clause form, or in other word called PROLOG form by following nine steps of syntatic manipulation. Let us consider the English statement:

"All persons who are clever or strong will win"

and convert it into the WFF.

$(\forall x)(\text{person}(x) \wedge (\text{clever}(x) \vee \text{strong}(x)) \Rightarrow \text{will\_win}(x))$.

The following steps are;

1) Eliminate implication with the equivalence.
   $(\forall x)(\neg(\text{person}(x) \wedge (\text{clever}(x) \vee \text{strong}(x))) \vee \text{will\_win}(x))$,

2) Reduce the scope of negation using De Morgan's Laws.
   $(\forall x)((\neg\text{person}(x) \vee (\neg\text{clever}(x) \wedge \neg\text{strong}(x))) \vee \text{will\_win}(x))$,

3) Standardise the variables so that the local variables within the quantification become unique. Example;
   $(\forall x)\neg P(x) \vee (\exists x)Q(x) \Leftrightarrow (\forall x)\neg P(x) \vee (\exists y)Q(y)$,

4) Eliminate the existential quantifiers using the skolem functions or skolem constants.
   a) $(\exists x)P(x) \Leftrightarrow P(b)$ where b is a skolem constant,
   b) $(\forall x)(\exists y)P(x,y)$ this is the same as $(\forall x)P(x,D(x))$ where $y = D(x)$ is a skolem function,

5) Move all the universal quantifiers to the front. Example;
   $(\forall x)(P(x) \wedge (\exists y)Q(y) \Rightarrow Z(x, y)) \Leftrightarrow$
   $(\forall x)(\exists y)(P(x) \wedge Q(y) \Rightarrow Z(x, y))$,

6) Convert to the conjunctive normal form (make up into the conjunction clause).
   $(\forall x)((\neg\text{person}(x) \vee \neg\text{clever}(x) \vee \text{will\_win}(x))$
   $(\neg\text{person}(x) \vee \neg\text{strong}(x) \vee \text{will\_win}(x)))$,

7) Delete all the universal quantifiers. From now on all variables will be assumed to be universal quantified.
   $((\neg\text{person}(x) \vee \neg\text{clever}(x) \vee \text{will\_win}(x))$
   $(\neg\text{person}(x) \vee \neg\text{strong}(x) \wedge \text{will\_win}(x)))$,

8) Split into the clause form by eliminating the "top level" conjuction.
   $(\neg\text{person}(x) \vee \neg\text{clever}(x) \vee \text{will\_win}(x))$ and
   $(\neg\text{person}(x) \vee \neg\text{clever}(x) \vee \text{will\_win}(x))$,

9) Standardise the variables apart (i.e. making the variables local and unique).
   $(\neg\text{person}(x) \vee \neg\text{clever}(x) \vee \text{will\_win}(x))$ and
   $(\neg\text{person}(y) \vee \neg\text{clever}(y) \vee \text{will\_win}(y))$.

We can write in PROLOG form by converting back into the inplication form, if the above resulting clauses are HORN clause.

person(x) $\wedge$ clever(x) fi will_win(x) and
person(y) $\wedge$ strong(y) fi will_win(y)

That is the same as;

```
will_win(x) :- person(x), clever(x).
will_win(y) :- person(y), strong(y).
```

The point of all this is that the conversion process can be mechanised hence leading weight to the claim that PROLOG really is a programming in logic.

Database Viewed Through Logic

First of all, before considering the formalization of database in terms of logics, we have to mention some assumptions that given query (and integrity constraint) evaluation of databases. We define three such assumptions;

1) The closed world or convention for negative information which state that facts not know to be true are assumed to be false,
2) The unique name which states that individuals with different names are different,
3) The domain closure which states that there are no other individuals than those in the databases.

With reference the above assumptions, we can answer queries involving negation. For example the query like:

"Who is not a full time student"

addressed to a database whose current state consists of;

```
full_time_student(ahmad).
full_time_student(majid).
part_time_student(azmi).
part_time_student(ismail).
```

so, we will get an answer {azmi, ismail}. Here, the domain closure assumption restricts the individuals to be considered to the set of {ahmad, majid, azmi, ismail}. According to the unique name assumption, we have the following: ahmad not equal to majid, ahmad not equal to azmi, etc. Consequently, ahmad not subset of part_time_student, which according to the closed world assumption, leads to ¬part_time_student(ahmad).

A database can be considered from the viewpoints of logic in two different ways;

1) Interpretation.
   The queries and intergrity constraints are formulas that are to be evaluated on the interpretation by using the semantics definitions of truth,
2) Theory.
   The queries and the integrity constraints are considered to be theorems that are to be proved.

The use of logic for data description will abolish the distinction between the databases and programs, even the techniques apply to the problems in both fields. A strategy which applies to the execution of a program might also apply to the retrieval of answers to database queries. Methods for proving properties of program apply to verification of integrity constraints and retrieval of answers to queries.

The relationships between the relational query languages and user-interfaces have so many problems which can be transformed into logic. For instance;

1) A database system can be considered as a question-answering system. This is from the viewpoints of the theorem proving,
2) A database system can be viewed as an intelligent knowledge-based system if the queries can be written in very high level non-procedural query language to perform certain tasks,
3) A database can be viewed as a logic program, retrieval is automatically taken care of through resolutions. Because of the non-distinction between input and output, any argument or combination of any argument can be chosen for retrieval,
4) The efficiency of access strategy directly effects the performance of database system.

Logic should have a positive effect on database design and usage as it provides a conceptual framework for expressing facts, integrity constraints and queries.

Interpretation Of Prolog

The execution of a PROLOG program can be considered as a search in an absract tree. The classical interpretation is from left to right and depth first search, checking one solution at a time. A more eficieny strategy for database access will be to globally process sets of solutions produced by every search (solution of goals). These solutions may, in turn, produce sets of goals (predicates to be verified), which can be globally verifies through filtering (i.e. the filter will be composed of four parts such as transfer controller, lexical automation, syntatic automation and result selection operator). Consider the following example:

```
z(X,Y) :- z1(X), z2(Y).
z1(a).
z1(b).
z2(c).
z2(d).
```

If we query the above example with ?–z(X, Y) then the search on z1 returns two solutions: X ← a and X ← b which in turn produce two sets of goals, there are z(a, Y) and z(b, Y). These two set of goals may be searched in a single pass on z2, through the backtracking and sequential processing. Finally, the first set of goal consists of {(X ← a, Y ← c) and (X ← a, Y ← d)} and the second set consists of {(X ← b, Y ← c) and (X ← b, Y ← d)}.

The concept of backtracking and cut (!) are explained more details in the standard text by Clockin W. F., et al 1981.

Unifications

A major difference between a PROLOG database and a relational database is the possibility of having variables in the data. The variables can be dependant and define relations between the argument. Unification is a process of finding a substitution of constant terms for variables to make expressions identical and check the substitutions for its consistency. It could be informally described as two-way or symmetrical matching, in as much as the distinction between the pattern and the data has been eliminated and variables can be bound on each side. It is possibly to unify more than two patterns at a time.

Not all variables need be instantiated (bound to constants) in the final identical expressions. The final expression is called the common instance or ground instance if all variable are instantiated. The minimal set of substitutions necessary to produce the common instance by applying the substitutions to any of the original pattern which is called most general unifier. The most general unifier is not unique but the common instance is.

Now, we give a unification procedure to find the most general unifier for any given clause P if P is a variable and results failure if P is not unifiable. The unification procedure makes use of two *"program variables"* R_k and k, which are initially set to $\in$ (i.e. empty substitution $\in = \{\}$) and 0 respectively. Through out these operations, the unification will alter the values. Thus k = 0 and $R_o = \in$. The eventual value of R_k is the most general unifier of the given clause P if P is unifiable and is $\in$ (i.e. not unifiable) if P is not unifiable. Below are the steps of the unification procedure;

1) If PR_k contains only one literal, then return R_k as the most general unifier and stop,

2) If PR_k contains more then one literal, find the first symbol position (see arrow below) for each literal in which not all literals have the same symbol. For instance:

   $PR_k = \{Q(\underset{\uparrow}{g}(x), a, f(u, v)) / Q(\underset{\uparrow}{u}, a, z)\}$.

   where "/" means substitutions,

3) Construct the disagreement set for PR_k, (i.e. contains the WFFs expressions (terms or literals) form each literal in PR_k that begin at the valued position). Disagreement set for the above example is {g(x)/u},

4) If there exist two terms s_k and t_k in the disagreement set such as that s_k is a variable symbol and t_k does not contains s_k, than take any two such terms s_k and t_k, replace R_k by $R_{k+1} = R_k\{(t, s)\}$ and replace k to k+1 and go to step 1. Form the above example: Let take s_k to be u and t_k to be g(x). Thus

   $R_{k+1} = R_k \{(G(x) / u)$

   and $PR_{k+1} = \{Q(g(x), a, f(u, v)) / Q(g(x), a, z)\}$.

5) If there do not exist two terms s_k and t_k in the disagreement set, then reports that P cannot the unified and stop.

No proof will be offered that the unification procedure does in fact find the most general unifier. With the above example if P were initially the clause {Q(g(x), a, f(u, v)), Q(u, a, z)} then the procedure would return the most general unifer $R_2 = \{(g(x), u), (f(g(x), v), z)\}$ and the most general unifier of P would be

$PR_2 = \{P(g(x), a, f(g(x), v)\}$.

PROLOG use the unification process for three important purposes;

1) To "decide which clause to invoke" (i.e. only the ones whose heads can be unified with the current goals),

2) To "pass the actual parameters to the clause". This can be achieved by substituting for the local variables to the corresponding terms found in the goal (call) pattern,

3) To "deliver the results of a clause" by computing terms to be substituted for the corresponding variables in the goal (call) pattern. It means that any numbers of the results can be returned and different invocation of the same clause.

Example Of Prolog + Relational Database And Query

PROLOG can be considered as a representing both relational algebra and relational calculus in a relational database. Every relational operator can be expressed in PROLOG and additional operators are available (e.g. implication) together with others facilities (e.g. list processing). Queries can be done by two operations;

1) Primitive operations.

Assume we have a relation schema which specifies names, domains and order or attributes, example;

```
STUDENT(integer matno, char name, integer schoolno, integer schship, char lecno).
student(12,AHMAD, 01,4500,L19).
student(21,NATHEN,02,3000,L12).
student(22,ISMAIL,03,4100,L42).
```

a) Selection.

The procedure selection can be used to generate all tuples for students of school 2 earning scholership over $2700.00.

```
select(Matno,Name,Scholership,Lecno);
    student(Matno,Name,2,Scholership,Lecno),
    Scholership > 2700.
```

b) Projection.

The procedure projection can be used to generate all the tuples and then project all the tuples with attributes name, matno and scholership,

```
project(Name,Matno,Scholership);
    student(Matno,Name,_,Scholership,_).
```

c) Join.

The procedure join is joining two relations (i.e. STUDENT and SCHOOL) and display all the tuples. Assume we have relation schema:SCHOOL (integerschno, charname, charlecno).

```
join (Matno, STDName, SCHno, Scholership, Lecno, SCHName);
    student (Matno, STDName, SCHno, Scholership, Lecno),
    school (SCHno, SCHName, Lecno).
```

d. Composition of two operations.

The procedure select_then_project is acomposition of two operations (i.e. select and project). Select student from school 1 and scholership over $3700.00 and project all tuples with atributes name, matno and scholership.

```
select_then_project(Name,Matno,Scholership);
    student(Matno,Name,1,Scholership,_),
    Scholership > 3700.
```

2) Set of operations are even more straight forward by applying the PROLOG language to the relational database. Assume we have two relations schema (i.e. a and b) with n-arys.

```
a(Y1,Y2,..., Yn).
b(Y1,Y2,..., Yn).
```

The primitive operations are:

i) Union.

```
a_UNION_b (Y1,Y2,...,Yn)        :-
        a(Y1,Y2,...,Yn); b(Y1,Y2,...,Yn).
```

ii) Intersection.

```
a_INTERSECTION_b (Y1,Y2,...,Yn). :-
        a(Y1,Y2,...,Yn), b(Y1,Y2,....Yn).
```

iii) Difference.

```
a_DIFFERENCE_b (Y1,Y2,...,Yn)  :-
        a(Y1,Y2,...,Yn), not b(Y1,Y2,...,Yn).
```

All of the query operations can be explained in terms of static interpretation of procedures (i.e. predicate). Queries which involve only primitive operations can be answered without actually creating the resulting relation. Its tuples can be generated by a failure-driven loop and displayed immediately. We also can use a "bagof" predicated and sometime its more efficient. Let see the commands below:

a) "bagof" command.

We look for the maximum scholership from relation STUDENT.

```
:-  bagof(Scholership, student(_,_,_,Scholership,_), SCH), maximum(SCH,STDScholer),
    write(STDScholer), nl.
```

b) Failure-driven.

We look for student of school 2 who earns not more than $5000.00 and who are members of AICLUB since at least 1980. Assume we have ralation schema: AICLUB(integer matno, char name, integer status, integer datejoined).

```
:-  student(Matno,Name,2,Sno,_),
    Sno = < 5000m
    aiclub(Matno,_,_,Datejoined),
    Datejoined = < 1980,
    write(Matno,Name), nl,
    fail.
```

Sometimes the queries cannot be expressed in terms of a composition like the above operations so we must use the concept of programming in logic and recursion. The main advantages of using PROLOG is that these queries can be represented in terms of tuples of facts and rules.

Conclusions And Future Works

On the whole, PROLOG is a powerful tool for database applications. However the saiz of a real database may far exceed the capacity of any existing PROLOG implementation. Suprisingly, PROLOG is in a sence, too strong and too unrestricted. Unrestrained use of "assert/retract" may be also run the integrity of a database.

Consequently, PROLOG should rather be considered a tool for implementing more restricted user interfaces; queries and commands in a user language are analysed (types checked, integrity ensured, etc) and only then translated into PROLOG.

It has been designed and used with the software engineer "mind" rather than computer. It also allow the engineer to set up a precise definition of what the problem is and it enables the computer to apply the fundamental logic or logical reasoning to database problem.

We have atempted to explain how mathematical logic provides a conceptual framework for database systems, in particular we have shown how it is use to represent and manipulate facts, formulate and evaluate queries. Indeed, logic provides an appropriate framework for many database system. For example in previous section we saw how programming in logic, relational database system and query are closely interrelated. The following is a list of research and development (R&D) topics;

1) Research on parallel processing for optimizing the disc access: distribution of search on several disc units, parallel verification of clause (OR-parallelism) (i.e. several clauses matching a goal are processed concurrently). Parallel verification of independent literals in a single clause (AND-parallelism) (i.e. several goals in a clause are executed concurrently),
2) Designing a natural language query system for database,
3) Optimizing query evaluation which is based on the semantic knowledge (i.e. interactive access to the database and in natural language context),
4) The efficiency of the algoritham (i.e. already developed) for executing a disributed PROLOG program on a broadcast network,
5) Embedding date manipulation languages in programming languages. It is possible to interface a DBMS with a logic programming language such as PROLOG.
6) There are many new types of databases which may be considered for special treatment in the future;
   a) Numeric database
      The main problem is the representation of vast and often sparse matrice. This database is for compact storage and selective usage,
   b) Text database
      This database deals with textual information, such as in document retrieval system and word–processing environment,
   c) Image database
      This database is used to store the images/pictures of objects using either graphic and/or digitisation such as storage of thumb prints,

d) Speech database

Sound would be stored, analysed and selectively combined and reproduced. This should also allow operations and conversion of sound into text.

Logic and database systems are the core research in "Japanese Fifth Generation Project". Lastly, logic provides a firm theoretical basis upon which are can purpose database theory in general. There are many research areas that remain to be investigated in addition to those listed above.

References

1) Cambell J. A., *Implementation Of PROLOG*, Ellis Horwood Series, 1984.

2) Clockin W. F. and Mellish C. S., *Programming In PROLOG*, Springer-Verlage, 1981.

3) Chao-Chin Yang, *Relational Databases*, Prentice-Hall Int. Series, 1986.

4) Date C. J., *An Introduction To Database System*, Addison-Wesley Pub. Co., 1981.

5) Gallaire H. and Minker J., *Logic and Database*, Addison-Wesley Pub. Co., 1977.

6) Gray P., *Logic, Algebra and Database*, Ellis Horwood Series, 1984.

7) Inria F. B., *International Conference On Very Large Database*, Proc. of IEEE Computer Society, ACM and Canadian Information Processing Society, 1980.

8) Knuth K. D., *The Art Of Computer Programming: Fundamental Algorithms*, Vol. 1, Addison-Wesley Pub. Co., 1979

9) Kowalski R., *Predicate Logic As A Programming Language*, Ellis Horwood Series, 1974.

10) Nilsson N. J., *Principle of Artificial Intelligence*, Springer-Verlage Pub., 1982.

11) Warren H. D., Pereira L. M. and Pereira F., *"PROLOG: The Language And Its Implementation Compared With LISP"*, Proc. of ACM Symposium On AI Programming Language, 1977.