International Journal of Computer Applications (0975 – 8887)
Volume 71– No.24, June 2013

Development of Prolog for Database Management System

Pradeep Kumar
Research Scholar
SSJ Campus, Almora
Kumaun University

Sumit Khulbe
Teaching Personnel,
SSJ Campus, Almora
Kumaun University

Prof. H S Dhami
Dept. of Mathematics,
SSJ Campus, Almora
Kumaun University

ABSTRACT
The present paper aims at the development of logical programming language (Prolog) for implementing database concepts in the form of establishment of links between logic programming and databases. We have been able to implement a deductive/inference database management and Knowledge-based systems in Prolog.

Keywords
Prolog mechanism, Indexing, Knowledge-based systems and Deductive/Inference model.

1. INTRODUCTION
We know that PROLOG is one of the most popular languages based on an inference mechanism and many expert systems are implemented in PROLOG. In the present work, we have selected this language to express the rules and a PROLOG interpreter and it's uses as an inference mechanism, taking the initiative from the work of A Domenici et al. [1]. We have also utilized the work of Pascual Julián-Iranzo et al. [16], who have examined the extension of Prolog in order to be able to deal with similarity-based fuzzy unification and the work of R.J. Lucas [18], who has kept the options open to implementers of Prolog interfaces to relational databases.

While interest in the use of Prolog for database applications is growing, the size of such applications is limited on account of the ability of current implementations of Prolog for handling disc-resident clauses. Various software and hardware approaches, such as codeword indexing, have been put forward to solve this problem, S Zhou et al. [20] have examined the reports of the comparative performances of one-level and two-level codeword indexing. In another research work F. Gozzi et al. [8], have described the PRIMO system as a PROLOG-Relational Interface. The main design goals of PRIMO include portability, modularity, and transparency. As a result of portability, the PRIMO interface can be established between any two PROLOG and relational systems; provided that the former supports a call to the operating system and that the latter supports SQL.

In Prolog database space, the cache architecture was discussed by Lanfranco Lopriore [11]. In this paper Lanfranco has reduced the database memory requirements of Prolog programs used in DBMS. N.W. Paton et al. [14] have used Prolog to implement in object-oriented database. Research on Prolog – based object oriented engineering in DBMS, present the primary concepts of PBASE, a prototype object oriented database system. PBASE is intended to support the needs of engineering applications with specific reference to structural engineering, as mentioned by A.S. Watson et al. [2]. Another work in prolog and a rational DBMS for decision support systems was given by Jorge Bocca et al. [10]. Research on techniques for implementation of the system which is implemented by coupling Prolog with a commercial relational DBMS will be a useful tool for designing expert systems, especially for designing expert systems that have a requirement for knowledge-directed processing of large amounts of shared information given by Yuguo Zhang et al. [23]. A compendium of research on PROLOG with database management for different systems has been given by Deyi Li [6].

Research on Applying Prolog to Develop Distributed Database Systems has been conducted on a number of different distributed databases, given by Nuno P. Lopes et al. [15]. Implementation of an integrated multi-database for PROLOG systems have been covered as part of large systems by DA Bell et al. [5]. A table oriented database-prolog system approach given by S.M Kuo Pan et al. [19].

In this paper, we investigate deductive inference for interiors and exteriors of Horn knowledge bases, where interiors and exteriors are those which were introduced by Makino et al [12] to study stability properties of knowledge bases for PROLOG systems. Knowledge-based systems are commonly used to store the sentences as our knowledge for the purpose of having automated reasoning such as deduction applied to them given by Brachman et al [3]. Deductive inference is a fundamental mode of reasoning, and is usually abstracted as follows, as given in the work of Makino and Ono [13]:

Given a knowledge base KB, assumed to capture our knowledge about the domain in question, and a query χ that is assumed to capture the situation at hand, decide whether KB implies χ, denoted by KB $\models \chi$, which can be understood as the question: "Is χ necessarily true given the current state of knowledge?"

In this paper, we consider interiors and exteriors of knowledge bases used by PROLOG database.

We have also used the Deductive/Inference model for conclusion based on the form of premises interpreted as a function. In order to provide rules of inference with a sufficient computational power, the logical Data Model also supports a concept of so-called computational terms. Primitively speaking, computational terms are arithmetical expressions having free variables, constants and/or other terms as parameters. The following notation is used to define such computational terms:

([Parameter_1] [Operation] [Parameter_2])

Where [Parameter_1] and [Parameter_2] are free variables, constants or other computational terms, and these terms are used as free variables in rules of inference for Query analyzed.

For this concept we have derived a method for query evaluation in deductive databases and it is based on discovering of axioms and facts relevant to a given query. Such type of work had been proposed and generated by Eliezer L. Lozinskii [7]. We have also modified the Query evaluation and optimization technique of Wei Lu [22] in deductive and object-oriented spatial databases. It has been demonstrated that our system is able to improve the quality of the state of Prolog systems. Our approach can be regarded as converting the more common way of using a Prolog system to automatically post-edit the output of DBMS systems by Deductive/Inference model, as described in Troels Andreasen [21]. Similar work references can be seen in the works of Chan Chee keong et al. [4]) for Implementation of a deductive database system using SQLBase and in Prolog Programming Language Encyclopedia of Physical Science and Technology in DBMS verification, given by Heimo H [9]. In this paper we are presenting a methodology as a Knowledge base system, introduced by R.J. Gil [17] for a novel integrated knowledge support system.

2. METHODOLOGY

Prolog automates and reduces much of the daily construction paperwork process while associating pertinent related information. Prolog offers clear visibility into what is happening on a project to the client, site and home office project management. This allows for closer collaboration and improved communication between home office and field personnel, so for these facts and findings we used the concept of Prolog clauses as follows:-

To find a clause that matches the query:-

?- F(a,b), the Prolog system does not look at all the clauses in the knowledge base only the clauses for f. Associated with the function f is a pointer or hashing function that sends the search routine directly to the right place. This technique is used in this paper and also called as indexing.

Many implementations carry this further by indexing on not only the predicate, but also the principal function of the first argument. In such an implementation, the search considers only clauses that match F(a,?) and neglects clauses such as F(b,c).

Indexing can make a predicate Head/Tail recursive when it otherwise would not be. For example,

F(x (A, B)):- F (A).

F (q).

It is tail recursive even though the recursive call is not in the last clause, because indexing eliminates the last clause from consideration: any argument that matches x (A, B) cannot match q. The same is true of list processing predicates of the form

F ([Head | Tail],?) = F ([],?)

Here [] means the null array where user can formed any type of string operation of database, to reduce the cost.

The standard case, the simplest one, is the case where the goals of the query refer to non-recursive virtual predicates, each predicate being defined only by one clause. Most of the PROLOG-DBMS presented in this paper reduce the clustering of data used.

In fact this general methodology is modified when a predicate definition contains:-

Evaluable predicate, or

Negation, or

A “cut", or

A predicate defined by several clauses, or

A recursively defined predicate.

Query is terminated, the corresponding sub-goals are replaced in the PROLOG program by a new predicate which is used to represent the ground clauses associated to the query answer, and the resolution of data in database is continued. We are working in a structure (D, F) and let V be a universal set of variables, given once and for all, used to refer to the elements of its domain D . We will assume that V is infinite and countable. We can now construct syntactic objects of two kinds, terms and constraints. Terms are sequences of DBMS elements from

V~F of one of the two forms of a particular data,

x or f t 1 .....t n, where x is a data variable, f is an n-place operation and where the ti are less complex terms (i=1…..n).

In this approach the number of call to the DBMS can be quite important, leading to an important overhead, because for each call the DBMS has to analyze the query, to optimize it, and so on. But the advantage is that the size of each answer is much smaller.

3. DESIGNING TOOLS USED IN QUERY OPTIMIZATION

Design is a process by which design intentions are transformed into design descriptions and has identifiable phases or sub processes. Although the phases may not be addressed hierarchically for the entire design cycle and are often carried out recursively, there is an inherent order in the way in which designers approach a design problem, the following represents one decomposition.

(i) Design analysis or formulation involves identifying the goals, requirements and possibly the vocabulary relevant to the needs or intentions of the designer. It is the development of the detailed specification of the design brief,

(ii) Design synthesis involves the exploration of a design space, producing a solution includes the formation or selection of a prototype followed by elaboration or refinement.

(iii) Design evaluation involves interpreting a partially or completely specified design description for conformance with goals and/or expected performances.

For every stored relation (R) with an attribute (A) we keep:

•|R|: the cardinality of R (the number of tuples in R)

• b(R): the blocking factor for R

• min(R; A): the minimum value for A in R

• max(R; A): the maximum value for A in R

• distinct (R; A): the number of distinct values of A

We use the following insertion sort program to illustrate how simple Prolog programs are typically constructed.

```
% insertion sort (simple version)

isort([], []).

isort(A.As, Bs) :-

isort(As, Bs1),

insert(A, Bs1, Bs).
```

The isort predicate follows a typical pattern for list processing. There is a base case for [] and a case for A.As with a recursive call containing as and another call containing A. The structure of insert is similar, except the case dealing with non-empty lists has been split further into two sub-cases. The two basic techniques used are structural induction and case analysis. A typical programmer using a modern functional language would code insert in a similar way to the Prolog version, using structural induction and case analysis. The code for isort may also b e written in the same style. However, a good functional programmer should realize that the resulting code follows a familiar pattern. That pattern is generalized by the higher order function foldr, which can be used to code isort more simply as follows:-

```
isort As = foldr insert [] As
```

As described in Fig1 :

4. FRAMEWORK OF PROLOG IN DBMS

We have presented an Interface based on the static approach. It is interesting to compare its advantages and drawbacks with regard to the dynamic approach. In the following we do this comparison taking into account two different criteria: the number of interactions between the PROLOG interpreter and the DBMS, and the size of the answers obtained from the DBMS which must be stored in core memory. The differences are significant only in the case of the queries whose evaluation involves predicates which are recursively defined.

To design the software module, called Framework, which interfaces the PROLOG interpreter and the Relational DBMS we adopted the following guidelines in this paper:-

1. The queries sent to the DBMS should correspond to the largest sub-sets of the PROLOG program which can be independently evaluated by the DBMS.

2. The queries are not transformed by the Framework in order to optimize query evaluation; this optimization is supposed to be done by the DBMS.

3. Neither the Prolog interpreter, nor the DBMS modules are modified; the consequence of this requirement is that the Interface does not need to know how these two modules are implemented.

We attempt to propose an extensible framework to database management system (DBMS) which deals with the unique and emerging challenges for Prolog or any other language as depicted here a data. We addressed extension required to various existing components like data types, data operations, indexing, data integration, data storage and management in order to process and manage Prolog efficiently.

```
remove_duplicates([First|Rest], Result) :-

member(First, Rest),!,

remove_duplicates(Rest, Result).

remove_duplicates([First|Rest], [First|New_Rest]) :-

% First does not occur in Rest.

remove_duplicates(Rest, New_Rest)

sum_list(Number_List, Result) :-

sum_list(Number_List, 0, Result).

% sum_list(+Number_List, +Accumulator, ?Result)

sum_list([], A, A). % At end: unify with accumulator.

sum_list([H|T], A, R) :- % Accumulate first and recur
```

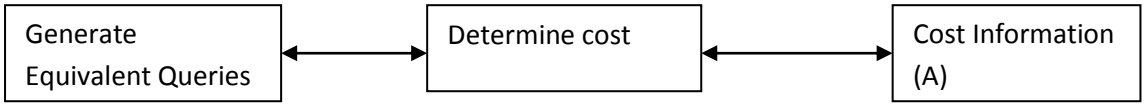

Fig 1:

5. MODIFICATION OF THE FRAMEWORK

In this research paper we have used the concept of Knowledge-based system for truth values which was found in a article of Brown et al. 2003. Knowledge-based system is commutative, as in that $x \vee y = y \vee x$ for disjunction and $x \wedge y = y \wedge x$ for conjunction. In our present research work , we have attempted x→user input, and y→system output. If the user wants to check that whether the database is correct for user choice then he/she has to keep this fact into mind that the leading database management systems like Oracle, IBM DB2, Microsoft SQL Server, Metadata are equipped with complex data types which can store or cannot be store/support any Prolog data types which are normally sequences, primary and secondary structures, and images. In such cases they are supposed to provide object oriented development framework and direct support to store and process XML which are highly recommend to advanced Prolog data types. They are powerful enough to support data quality, integrity, availability, security, manageability, interoperability and an ability to manage and process huge volumes of data.

Again we use recursive functions to correct the database errors and formulated them with respect to the following three rules and apply them to understand in database through language:-

1. a V (b V c) = (a V b) V c ----------→associative

2. a V b= b V a ----------→commutative

3. a→b ➔ b→c then a→c ----------→transitivity

6. DESCRIPTION OF THE PROJECT

```
% remove_duplicates(+List,-ProcessedList)

% Removes the duplicates in List, giving Processed List.

% Elements are considered to match if they can

% be unified with each other; thus, a partly uninstantiated

% element may become further instantiated during testing.

% If several elements match, the last of them is preserved. public class SuffixTree {  public void sample Usage() {

    %:sum_list(+ListOfNumbers,?Result)

%. Sums the numbers in the list, giving Result. Crashes with an

%. error message if first argument is not a list of numbers.

sum_list([],0). % Empty list sums to 0.

sum_list([First|Rest],N) :- % Add first number to sum of rest.

number(First),!,

sum_list(Rest,R),

N is First+R.

sum_list(_,0) :- % Catch ill-formed arguments.

errmsg('First arg of sum_list/2 must be a list of numbers.').

class CompactSuffixTree extends AbstractSuffixTree {

  public CompactSuffixTree(SimpleSuffixTree simpleSuffixTree) {

    super(simpleSuffixTree.text);

    super.root = compactNodes(simpleSuffixTree.root, 0);

    super.best = simpleSuffixTree.best;

  }  private SuffixTreeNode compactNodes(SuffixTreeNode node, int nodeDepth) {

    node.nodeDepth = nodeDepth;

    for (SuffixTreeNode child : node.children) {

      while (child.children.size() == 1) {

        SuffixTreeNode grandchild = child.children.iterator().next();

        child.incomingEdge.label += ", "

            + grandchild.incomingEdge.label;

        child.stringDepth += grandchild.incomingEdge.label.length();

        child.children = grandchild.children;

        // for (SuffixTreeNode grandchild : child.children)

        grandchild.parent = node;       }

      child = compactNodes(child, nodeDepth + 1);       } return node;  }}

class SuffixTreeNode {

  AbstractSuffixTree tree;

  SuffixTreeEdge incomingEdge = null;

  int nodeDepth = -1;

  int label = -1;

  Collection<SuffixTreeNode> children = null;

  SuffixTreeNode parent = null;

  int stringDepth;

  int id = 0;

  public static int c;

  public int visits = 1;

  public SuffixTreeNode(AbstractSuffixTree tree,

   SuffixTreeNode parent,

    String incomingLabel, int depth, int label, int id) {

    children = new ArrayList<SuffixTreeNode>();

    incomingEdge = new SuffixTreeEdge(incomingLabel, label);

    nodeDepth = depth;  this.label = label;  this.parent = parent;

    stringDepth = parent.stringDepth + incomingLabel.length();

    this.id = id;  this.tree = tree;

  }

public SuffixTreeNode(AbstractSuffixTree tree) {

    children = new ArrayList<SuffixTreeNode>();

    nodeDepth = 0;

    label = 0;

    this.tree = tree;  }

  public void addSuffix(List<String> suffix, int pathIndex) {

    SuffixTreeNode insertAt = this;

    insertAt = search(this, suffix);

    insert(insertAt, suffix, pathIndex);  }

  private SuffixTreeNode search(SuffixTreeNode startNode, List<String> suffix)  if (suffix.isEmpty()) {

      throw new IllegalArgumentException(


          "Empty suffix. Probably no valid simple suffix tree exists for the input.");

    }

    Collection<SuffixTreeNode> children = startNode.children;

    for (SuffixTreeNode child : children) {

      if (child.incomingEdge.label.equals(suffix.get(0))) {

        suffix.remove(0);

        child.visits++;

        if (child.visits > 1

            && child.stringDepth > tree.best.stringDepth) {

          tree.best = child;       }
```

```
            if (suffix.isEmpty()) {                    return child;
}
            return search(child, suffix);          }       }
      return startNode;    }
   private void insert(SuffixTreeNode insertAt, List<String>
suffix,
 int pathIndex) {
      for (String x : suffix) {
         SuffixTreeNode child = new SuffixTreeNode(tree,
insertAt, x,
               insertAt.nodeDepth + 1, pathIndex, id);
         insertAt.children.add(child);
         insertAt = child;       }    }
   public String toString() {
      StringBuilder result = new StringBuilder();
      String incomingLabel = this.isRoot() ? "" :
this.incomingEdge.label;              for (int i = 1; i <=
this.nodeDepth; i++)
         result.append("\t");       if (this.isRoot()) {
 c = 1;
         this.id = 1;
      } else {
         this.id = c;
         result.append(this.parent.id + " -> ");
         result.append(this.id + "[label=\"" + incomingLabel +
"\"]" + "("
               + visits + "," + (stringDepth) + ")" + ";\n");       }
      for (SuffixTreeNode child : children) {
         c++;
         child.id = c;
         result.append(child.toString());       }
      return result.toString();
   }    public String printResult()
}
```

7. BACKGROUND OF THE WORK

The concept of generating program source code by means of a dialogue involves combining strategies with system and user initiative. The strategy with system initiative safely navigates the user, whereas the strategy with user initiative enables a quick and effective creation of the desired constructions of the source code and collaboration with the system using obtained knowledge to increase the effectiveness of the dialogue.

The present invention sets forth a method and an arrangement for different database correction processing and can be automate the process of adapting domain specific Prolog understanding. It solves the problem of simple natural language understanding and allows users to interact with machines using natural language. This work shall be of immense importance to the students of programmers and DBA who sometime feel harassed while selection of database and moreover are not certain about the exercises in extraction of languages to develop a database.

8. CONCLUSION

We have presented the implementation of a PROLOG and DBMS interface which does not need any modification of the PROLOG interpreter or the DBMS evaluator. This interface, even if it is not optimal, allows clustering many calls to the DBMS since it can generate queries containing: AND, OR, or NOT operators, and existential quantifiers. We have suggested an increasingly active role for such systems which is likely to have the potential to change the design process itself. The logic programming community should take advantage of the higher order style of programming developed by the functional programming community. This style encourages more abstraction, more reuse of code and more concise data programs in database management systems. We have shown by example how higher order programming also fits well with the additional strengths of logic programming such as multiple modes and Meta data logics. To study such structures, especially to check recent database and to search for solutions may be made faster using a computer.

9. REFERENCES

[1] A Domenici, B Lazzerini, CA Prete (1990), Introduction to Prolog computation model and its implementation Information and Software Technology, Volume 32, Issue 6, and pp. 423-431.

[2] A.S. Watson, S.H. Chan (1991), A prolog-based object oriented engineering DBMS Original Research Article Computers & Structures, Volume 40, Issue 1, and pp. 11-21.

[3] Brachman, R. J. and Levesque, H. J. (2004), Knowledge Representation and Reasoning.

[4] Chan Chee Keong, Chen Yin (1997), Implementation of a deductive database system using SQLBase , Volume 20, Issue 6, and pp. 317-323.

[5] DA Bell, JB Grimson, DHO Ling (1989), Implementation of an integrated multi database-PROLOG system , Volume 31, Issue 1, and pp. 29-38.

[6] Deyi Li (2004), A PROLOG database management, Volume 3, Issues 3–4, ISBN 0-86380-014-9, and pp.219.

[7] Eliezer L. Lozinskii (1992), Inference by generating in deductive databases Data & Knowledge Engineering, Volume 7, Issue 4, and pp. 327-357.

[8] F. Gozzi, M. Lugli, S. Ceri (1990), An overview of PRIMO: A portable interface between PROLOG and

relational database Information Systems, Volume 15, Issue 5, pp. 543-553.

[9] Heimo H. Adelsberger (2003), Prolog Programming Language Encyclopedia of Physical Science and Technology in DBMS verification, and pp. 155-178.

[10] Jorge Bocca, Europeean (1986), EDUCE: Prolog and a rational DBMS for DSS, Volume 2, Issue 3, and pp. 274.

[11] Lanfranco Lopriore (1993), A data cache for Prolog architectures Future Generation Computer Systems, Volume 9, Issue 3, and pp. 219-234.

[12] Makino, K. and Ibaraki, T. (1996), Interior and exterior functions of Boolean functions. Discrete Applied Mathematics, Volume 69, and pp. 209–231.

[13]Makino, K. and Ono, H. 2011. Deductive Inference for the Interiors and Exteriors of Horn Theories. ACM Trans. Comput. Logic V, N, Article A, A.1-A.16..

[14] N.W. Paton, S Leishman, S.M. Embury, P.M.D Gray (1993), On using Prolog to implement object-oriented databases Information and Software Technology, Volume 35, Issue 5, and pp. 301-311.

[15] Nuno P. Lopes , Juan A. Navarroy , Andrey balchenkoy , and Atul Singh (2003), Applying Prolog to Develop Distributed Systems.

[16] Pascual Julián-Iranzo, Clemente Rubio-Manzano, Juan Gallardo-Casero (2009), Bousi~Prolog: a Prolog Extension Language for Flexible Query Answering in Theoretical Computer Science, Volume 248, and pp. 131-147.

[17] R.J. Gil, M.J. Martin-Bautista (2012), a novel integrated knowledge support system based on ontology learning: Model specification Knowledge-Based Systems, Volume 36, and pp. 340-352.

[18] R.J. Lucas (1991), Prolog—relational database interfaces Information and Software Technology, Volume 33, Issue 10, and pp. 734-740.

[19] S.M. Kuo Pan, Y. Kaneda (1989), A table oriented database-prolog system Micro processing and Microprogramming, Volume 25, Issues 1–5, and pp. 9-14.

[20] S Zhou, MH Williams (1991), Assessment of two-level codeword indexing applied to Prolog database Information and Software Technology, Volume 33, Issue 2 and pp. 157-162.

[21] Troels Andreasen (2003), An approach to knowledge-based query evaluation, Fuzzy Sets and Systems, Volume 140, Issue 1, and pp. 75-9.

[22] Wei Lu, Jiawei Han (1995), Query evaluation and optimization of in deductive and object-oriented spatial databases, Volume 37, Issue 3, and pp. 131-143. [23] Yuguo Zhang, Peter Hitchcock (1990), Coupling Prolog to a Database Management System , Volume 15, Issue 6, and pp. 663-667.