Malaysian Journal of Computer Science, Vol. 10 No. 1, June 1997, pp. 47-52

# AN EFFECT OF SIMPLIFYING MAGIC RULES FOR ANSWERING RECURSIVE QUERIES IN DEDUCTIVE DATABASES

***Ali Mamat***
Department of Computer Science
Universiti Putra Malaysia
43400 UPM Serdang
Malaysia
Tel.: 03-9486101x3695
email: ali@fsas.upm.edu.my

***Mustafa Mat Deris***
School of Information Technology
Kolej Agama Sultan Zainal Abidin
21000 Kuala Terengganu
Malaysia
Tel.: 09-6664466
email: mustafa@rs520.kusza.edu.my

***ABSTRACT***

*The basic magic sets transformation algorithm for rewriting logical rules in deductive databases is very clear and straightforward. However, rules generated by the algorithm for answering queries are too many compared to the original rules. Therefore, it is useful to simplify the generated rules before they are evaluated. This paper reports the study on the effect of simplifying such rules from the aspect of computing time. It is concluded that the improvement as a result of simplification is quite significant.*

***Keywords**: **deductive databases, magic sets method, rule/goal graph, magic rules***

## 1.0 INTRODUCTION

A deductive database consists of a set of rules and facts. A rule is of the form $A \neg B_1,\ldots, B_m$ where $A$, $B_1$, ..., $B_m$ are atomic formulas (atoms). $A$ is called the head of the rule and $B_1, ..., B_m$ is the body of the rule. Each $B_i$, $i=1, \ldots ,m$ is also called a subgoal. An atomic formula is of the form $p(t_1, \ldots , t_n)$ where $p$ is a predicate symbol and $t_1, \ldots , t_n$ are terms. Predicates represent database relations and they are classified into two types, namely, extensional database (EDB) predicates and intensional database (IDB) predicates. IDB predicates appear in the head of rules as well as in the body whereas EDB predicates appear only in the body of rules. Thus, EDB predicates correspond to database facts (tuples in relational databases). A rule is called recursive if the same predicate symbol appears both in the head and body of the rule.

In general, methods of answering queries in deductive databases may be classified into two classes, namely, top-down and bottom-up methods. Bottom-up methods are claimed to have advantages over top-down methods. In particular, the former return all answers to a query and do not go into an infinite loop. In other words, the computation terminates immediately after all answers to the query are returned. One of the bottom-up methods is magic sets [1]. In this method, the original rules relevant to a given query are transformed to a set of rules called magic rules. The transformation is guided by binding (a constant in the query) propagation behaviour during the top-down phase. The resulting magic rules are then evaluated so as to return answers to the query.

There are various forms of magic sets transformation algorithms and the most basic one can be found in [2]. In this transformation, two new types of predicates are introduced, namely supplementary predicate *supi.j* (*i* and *j* are integers) and magic predicate *m.p* where *p* is a predicate. These two predicates model the constant (binding) propagation strategy of top-down methods. In fact, the two predicates will produce a set of values which are used to restrict the computation so that only relevant tuples are generated. For each rule $r_i$, having $k$ subgoals, we create a single rule for *supi.j* where *j=0,1,2,...,k-1* and *i* is an integer representing the rule number.

As a working example, consider the same generation rules of Fig. 1 below:

$r_1$: sg(X,X) ← person(X)
$r_2$: sg(X,Y) ← par(X,XP),sg(XP,YP),
par(Y,YP)

Fig. 1: Rules for the same generation

The first rule says that every person is of the same generation as her/himself and the second rule says that two individuals are of the same generation if their parents are. Suppose we give the query $\neg sg(c,X)$ which asks for all the individuals of the same generation as *c*. The magic rules resulting from the above rules using the basic magic set transformation algorithm of Ullman [2] are shown in Fig. 2.

Group I

(1) m.sg(XP) ← sup2.1(X,XP).

Group II

(2) sup1.0(X) ← m.sg(X).
(3) sup2.0(X) ← m.sg(X).

Group III

(4) sup2.1(X,XP) ← sup2.0(X), par(X,XP).
(5) sup2.2(X,YP) ← sup2.1(X,XP),sg(XP, YP).

Group IV

(6) sg(X,X) ← sup1.0(X), person(X).
(7) sg(X,Y) ← sup2.2(X,YP), par(Y,YP).

Group V

(8) m.sg(c).

Fig. 2: Magic rules for the same generation

Group I consists of rules for magic predicates, Group II for zeroth supplementary predicates, Group III for other supplementary predicates, Group IV for IDB predicates, and finally, Group V consists of only one rule called the initialisation rule. The above way of grouping rules will make the subsequent discussion easy.

The basic magic sets transformation is very clear and straightforward. However, rules generated by this transformation algorithm are too many compared to the original rules as seen in the previous example. This suggests the generated rules could be simplified before they are evaluated in order to get a better performance in terms of computing time. According to Ullman, the simplification does not affect the computing time significantly, but makes the rules clearer [3]. We will show that this statement is not quite correct.

The remainder of the paper is organised as follows. Section 2 shows how the simplified magic rules are directly generated from a rule/goal graph. Section 3 gives the sample data used in the computation. Section 4 presents the CPU time (in seconds) for answering queries and finally, Section 5 concludes the paper.

## 2.0 GENERATION OF SIMPLIFIED RULES USING A RULE/GOAL GRAPH

Before going any further, we need to introduce the concept of *binding pattern*. A binding pattern for n-ary predicate $p$ is a string $s$ of length $n$ of $b$'s and $f$'s, where $b$ stands for bound and $f$ stands for free. The binding pattern indicates which arguments of $p$ are bound and which are free. If the $i$th symbol of the pattern is $b$ (respectively $f$), then the $i$th argument of $p$ is bound (respectively free). For a predicate $p$ with a binding pattern $s$, we write $p^s$ to denote the binding pattern for $p$. Binding patterns for predicates in a set of rules are determined by the propagation of bindings (constants) in a given query during a top down phase.

The binding patterns of predicates in a set of rules and a query can be represented by a rule/goal graph [4]. The rule/goal graph has two types of nodes, namely rule nodes and goal nodes. Goal nodes represent adorned predicates (predicates with binding patterns) in the rules and the query, and rule nodes represent rule adornments (rule with bound and free variables). The rest of the details regarding the rule/goal graph are omitted and we refer interested readers to [2, 4]. The rule/graph for the same generation rules of Fig. 2 and the query ← *sg(c,X)* is shown in Fig. 3.

In [5], it was shown how magic rules were generated by using a rule/goal graph. Basically, a rule/goal graph is traversed in a depth-first search manner and for each node visited, a particular magic rule associated to that node is generated. Nodes of the rule/goal graph are associated to magic rules in the following way:

(a) the root node is associated with the initialisation rule (Group V rules),
(b) goal nodes with IDB predicates are associated with rules for magic predicates (Group I rules),
(c) rule nodes of the form $r_{i.0}$ are associated with rules for zeroth supplementary predicates (Group II rules),
(d) rule nodes of the form $r_{i.j}$, $j <> 0$ are associated with rules for other supplementary predicates (Group III rules), and
(e) rule nodes with only one successor (goal-node child) are associated with rules for IDB predicates (Group IV rules).

This way of generating magic rules can generate simplified magic rules directly if we give an extra effort whenever a rule node $r_{i.j}$, $j$ is an integer including 0, is accessed. This node corresponds to a rule for a supplementary predicate. Here, we must determine whether or not the rule corresponding to such node should be in the final set of simplified rules. Ironically, there is a simple way to determine that. We only need to examine its left child (goal) node. If this goal node contains IDB predicate, then the supplementary rule must be in the final set of rules, otherwise it is not. Although a supplementary rule is not in the final set, such a rule will temporarily be generated and it will be used in substituting a predicate in the body of other magic rules. It is useful to mention here that the rule/goal graph, used to help generating magic rules and at the same time simplifying it, is constructed only once.

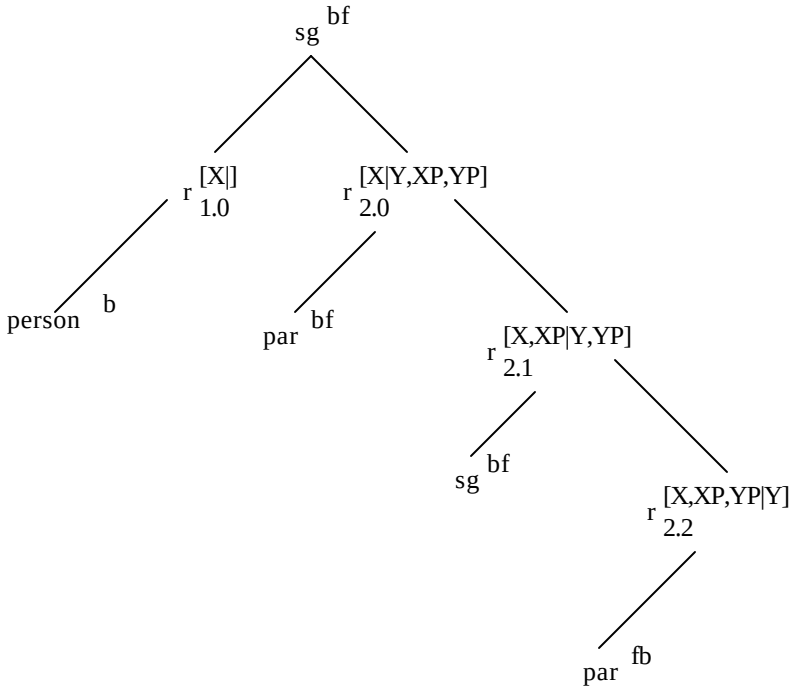


Fig. 3: Rule/goal graph for the same-generation rules

Let us now traverse the rule/goal graph of Fig. 3 in a depth-first search manner and as we traverse, we generate magic rules accordingly.

(a) The root node $sg^{bf}$

For this node, a rule for magic predicate *m.sg* is created. The argument of the predicate is the bound argument of the query. We thus have an initialisation rule

m.sg(c). (f1)

(b) Rule node $r_{1.0}$

A rule for zeroth supplementary predicate *sup1.0* is created. The argument of the predicate is the bound variable of the current node, namely *X*. The body of the rule consists of *m.sg* predicate, where predicate *sg* is predicate in the head of rule $r_1$ and the bound argument of the head becomes the argument of predicate *m.sg*. The resulting rule is

sup1.0(X) ← m.sg(X). (t1)

The above rule is not in the final set because the left child node of $r_{1.0}$ contains an EDB predicate. The next node to visit is the goal node $person^b$. Since the node contains an EDB predicate, there is no rule created for the node. The search then goes back to node $r_{1.0}$ and from there the search continues to the right child node. It happens that no such node exists, so we create one of the rules for IDB predicate *sg*:

sg(X,X) ← sup1.0(X), person(X).

Replacing predicate *sup1.0* by the body of rule (t1) above gives the rule

sg(X,X) ← m.sg(X), person(X). (f2)

(c) Rule node $r_{2.0}$

We next traverse the right branch of the root where we reach the rule node $r_{2.0}$. As with the node $r_{1.0}$, a zeroth supplementary rule is created, namely:

sup2.0(X) ← m.sg(X). (t2)

This rule is also not in the final set. The next node to visit is the goal node $par^{bf}$ but there is no rule associated to this node as it contains EDB predicate. A search then continues to the right child of node $r_{2.0}$, where node $r_{2.1}$ is reached.

(d) Rule node $r_{2.1}$

Initially, the supplementary rule generated for the current node is

sup2.1(X,XP) ← sup2.0(X), par(X,XP).

Notice that the left child node of the node in hand, i.e. node $sg^{bf}$, contains IDB predicate, hence the supplementary rule generated must be included in the final set of rules.

As with predicate *sup1.0*, predicate *sup2.0* is substituted by the body of rule (t2) and the result is

sup2.1(X,XP) ← m.sg(X), par(X,XP). (f3)

(e) Goal node $sg^{bf}$

We next consider the left child of node $r_{2.1}$, namely IDB goal node $sg^{bf}$ for which a magic predicate rule is generated:

m.sg(XP) ← sup2.1(X,XP). (f4)

Notice that here we do not need to replace the predicate *sup2.1* in the body of the rule because the rule for the predicate *sup2.1* itself was already included in the final set of rules.

(f) Rule node $r_{2.2}$

Like its parent node, we create for this node a rule defining a nonzeroth supplementary predicate. Variables of the predicate are *X* and *YP*. The resulting rule is

sup2.2(X,YP) ← sup2.1(X,XP), sg(XP,YP). (t4)

Finally, we also create a rule defining IDB predicate *sg* that appears in the head of rule $r_2$. The rule is

sg(X,Y) ← sup2.2(X,YP), par(Y,YP).

Replacing predicate *sup2.2* in the above rule by the body of rule (t4) results in the rule

sg(X,Y) ← sup2.1(X,XP),
sg(XP,YP), par(Y,YP). (f5)

As a summary, the resulting simplified rules are rules f1 to f5 as shown in Fig. 4, which are the same as those produced by the simplification algorithm of Ullman found in [2].

m.sg(c)
m.sg(XP) ← sup2.1(X,XP)
sup2.1(X,XP) ← m.sg(X), par(X,XP)
sg(X,X) ← m.sg(X), person(X)
sg(X,Y) ← sup2.1(X,XP),sg(XP,YP), par(Y,YP)

Fig. 4: The simplified magic rules for the same generation

## 3.0 SAMPLE DATA

Having simplified the magic rules, we need to investigate how much the simplification affects the computing time. In this respect, we will evaluate the sets of magic rules of Fig. 2 and Fig. 4 together with different sets of database facts by using the semi-naive evaluation method [6]. The data for *person* and *par* (parent) EDB predicates are based on the perfect binary tree shown in Fig. 5. The reason for choosing such structure is just to simplify data creation. The tree represents the family relationship where each node represents an individual and each branch represents a parenthood relationship. For instance, the root node $a_{11}$ and its child node $a_{21}$ form the fact (tuple) parent($a_{21}$,$a_{11}$). This fact means the parent of $a_{21}$ is $a_{11}$. The number of facts for *person* and *par* up to nth level of the tree are given by ($2^n$ - 1) and ($2^n$ - 2) respectively. Table 1 shows the number of facts of these two predicates for different levels of the tree.

Each row of Table 1 represents one set of database facts. For instance, when n=3 we have the following set:

person($a_{11}$).
person($a_{21}$).
person($a_{22}$).
person($a_{31}$).
person($a_{32}$).
person($a_{33}$).
person($a_{34}$).

par($a_{21}$,$a_{11}$).
par($a_{22}$,$a_{11}$).
par($a_{31}$,$a_{21}$).
par($a_{32}$,$a_{21}$).
par($a_{33}$,$a_{22}$).
par($a_{34}$,$a_{22}$).

The query given to each database (database facts plus magic rules) is of the form ← sg($a_{n1}$,X), where n is the lowest level of the tree. In other words, $a_{n1}$ is an individual represented by the left most node of the lowest level of the tree. To answer such a query, all *person* and *par* facts are involved in the computation.

## 4.0 RESULTS

The Central Processing Unit (CPU) time required to answer the query in the form of ←sg($a_{n,1}$,X), where n is the lowest level of the tree, against each database is recorded. Two readings have been taken as illustrated in Tables 2 and 3. Each table represents one reading. From the tables, it is clear that the readings are quite consistent in the sense that the improvement, as a result of the simplification, of the two readings is very close to each other. It can also be seen that the improvement decreases as the number of database facts (represented by the level of the tree) increases. As a matter of fact, most database processing usually deals with large volumes of data with thousands of tuples (facts). Even in such a situation, the

tables show that the effect of simplification still produces more than 10 percent improvement.

## 5.0 CONCLUSION

We have shown that the simplified magic rules can be generated directly by using a rule/goal graph. The approach used here is very clear, straightforward and the simplified rules are generated in one phase. It gives us the alternative way of simplifying magic rules for rule-based query evaluation. We have also shown that the resulting magic rules give better performance in terms of computing time - an observation that should not be neglected.

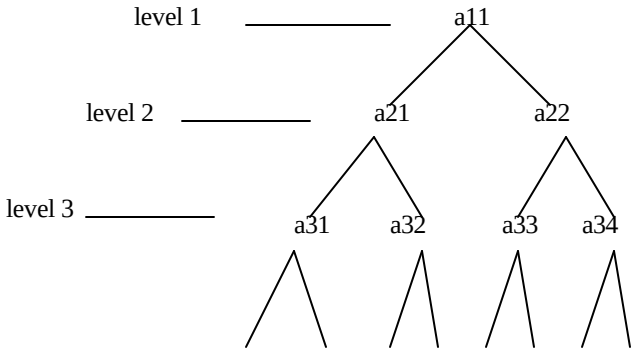


Fig. 5: A binary tree for representing a family relationship

Table 1: The number of facts for different levels of the tree

| Level of tree | No. of facts for person | No. of facts for parent | Total of facts |
|---|---|---|---|
| 3 | 7 | 6 | 13 |
| 5 | 31 | 30 | 61 |
| 7 | 127 | 126 | 253 |
| 9 | 511 | 510 | 1021 |
| 11 | 2047 | 2046 | 4093 |

Table 2: First CPU time reading

| Level of tree | Before the simplification (Seconds) | After the simplification (Seconds) | Improvement (%) |
|---|---|---|---|
| 3 | 0.073 | 0.056 | 23.28 |
| 5 | 0.336 | 0.259 | 22.91 |
| 7 | 1.835 | 1.548 | 15.64 |
| 9 | 12.700 | 10.715 | 15.62 |
| 11 | 110.158 | 98.791 | 10.32 |

Table 3: Second CPU time reading

| Level of tree | Before the simplification (Seconds) | After the simplification (Seconds) | Improvement (%) |
|---|---|---|---|
| 3 | 0.069 | 0.054 | 21.73 |
| 5 | 0.339 | 0.258 | 23.89 |
| 7 | 1.834 | 1.554 | 15.26 |
| 9 | 12.383 | 10.269 | 17.07 |
| 11 | 114.077 | 98.749 | 13.43 |

**REFERENCES**


[1] F. Bancilhon et al., "Magic sets and other strange ways to implement logic programs", in *Proceedings of the Fifth ACM Symposium on Principles of Database Systems, New York*, ACM, 1986, pp. 1-15.

[2] J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Vol. II*. Computer Science Press, 1989.

[3] J. D. Ullman, "Bottom-up beats top-down for datalog", in *Proceedings of the Eight ACM Symposium on Principles of Database Systems, New York*, ACM, 1989, pp. 140-149.

[4] J. D. Ullman, "Implementation of logical query languages for databases", *ACM Transactions on Database Systems*, Vol. 10, No. 3, 1985, pp. 289-321.

[5] A. Mamat, "A technique for transforming rules in deductive databases", *Pertanika Journal of Science and Technology*, Vol. 2, No. 2, 1994, pp. 121-136.

[6] F. Bancilhon, "Naive evaluation of recursively defined relation", in *M L. Brodie and J. Mylopoulos, (eds) On Knowledge Base Management Systems*, Springer Verlag, New York, 1986, pp. 368-378.


**BIOGRAPHY**

**Ali Mamat** is currently a lecturer in computer science at Universiti Putra Malaysia, Serdang. He obtained his Ph.D degree in Computer Science from the University of Bradford, U.K. in 1992. His research interests include databases and logic programming.

**Mustafa Mat Deris** is a lecturer in computer science at Kolej Agama Sultan Zainal Abidin, Kuala Terengganu. He obtained his M.Sc degree in Computer Science from the University of Bradford, U.K. in 1989. His research interests include computer performance and databases.

**ACKNOWLEDGMENT**

We would like to thank the referees for giving us good suggestions.