STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

SIMPLIFICATION OF MAGIC-SET RULES BY LOGIC GRAMMARS

D. TĂTAR AND V. VARGA

Abstract. The *magic sets* method is a bottom-up query evaluation technique that solves a query with particular *adornments* to the goal predicate. The program is transformed to an equivalent program using its adorned rules and *sideways information passing*. The transformed program has a bigger set of rules .The logic grammar as introduced in [6] can be used to simplify these rules.

1. Logic grammars

In [6] we defined a new concept of "Logic grammar" (LG) and we showed the soundness and completeness of them. In this section we shortly recall the basic concepts that are used in definition of "Logic grammar". Section 2 will present the magic-set method of optimization in Datalog programs [5] and in section 3 we will use LG's for simplification of magic-set rules. The language considered here is essentially that of the first-order predicate logic without function symbols. Let

- Pr be a set of predicates,
- C be a set of constants,
- V be a set of variables.

An atom over $C \cup V$ is of the form

$$p(u_1, \cdots, u_n), n \geq 0,$$

where $p \in Pr$ with arrity n, and each u_j is an element of $C \cup V$. If the arguments u_j are not interesting in a particular context, then we will denote an atom simply by p. Let A be the set of atoms over $C \cup V$, and, if $P' \subseteq Pr$, let $A_{P'}$ be the set of atoms with predicate symbols from P'. In some recent papers [1], [2], the set of predicates is considered as divided into two disjoint sets: the set EDB of extensional predicates (or extensional database predicates) which represent basic

Received by the editors: January 27, 1997.

1991 *Mathematics Subject Classification.* 68N17.

1991 *CR Categories and Descriptors.* D.1.6 [Programming Techniques]: Logic Programming; H.2.3 [Database Management]: Languages – data description languages, query languages.

"facts", and the set IDB of intensional databases predicates, representing facts deduced from the basic facts via the logic program [11]. Particularly, the set EDB can be the empty set (as, for simplicity, in most of the following demonstrations).

Definition 1.1. *A logic program P is a sequence of Horn clauses (definite clauses), that is, clauses of the form:*

$$p \leftarrow q_1, \ldots, q_n,$$

where p and $q_1, \ldots, q_n$ are atomic formulas in first-order logic, the comma is the logic operation "and", and the sign $\leftarrow$ is "if" or reverse of the logical implication.

We refer to the left (p) and right-hand side ($q_1, \ldots, q_n$) of a clause as its head and body. A clause is logically interpreted as the universal closure of the implication $q_1 \wedge \cdots \wedge q_n \rightarrow p$. If a clause has no right-hand side we will call it a fact or a unit clause. Let us observe that this definition considers only the class of positive logic programs (all atoms in all clauses are positive). From the properties of IDB and EDB predicates it follows that a predicate from the set EDB cannot occur in the head of clauses, but a predicate from the set IDB can occur in the set of facts.

Definition 1.2. *A goal G consists of a conjunction of atoms, and is denoted by:*

$$\leftarrow r_1, \cdots, r_t.$$

The following definition considers the fact that a goal G is treated by the resolution mechanism as a word rewritten by some well-defined rules. The last form of this rewriting tells us about some properties of the goal G.

Definition 1.3. *The logical grammar GL associated with a logic program P and the goal G is the system:*

$$GL = (I_N, I_T, X_0, F)$$

where:

- $I_N = A_{IDB} \cup \{X_0\}$ *is the set of nonterminals,*
- $I_T = A_{EDB} \cup \{\lambda\} \cup \{False, True\}$ *is the set of terminals,*
- X_0 *is the goal G,*
- F *is a finite set of production rules, of the form:*

$$p \rightarrow q_1 \cdots q_m, m \geq 1,$$

where $p \in IDB$ and "$p \leftarrow q_1, \ldots, q_m$" is a clause in the program P.

or

$$p \rightarrow \lambda$$

where "$p.$" is a unit clause in the program P.

We assume, in the following, that substitutions, composition of substitutions, and the most general unifier $\sigma = \mathbf{mgu}(g, h)$ of atoms g and h are defined as in logic programming, [1,2].

For a logic grammar GL we define the rewriting relation "$\Rightarrow$" as follows:

Definition 1.4. *If $R \in A^+$ and $Q \in A^*$, then*

$$R \Rightarrow^{\sigma}_{GL} Q$$

if there exists an atom $h \in I_N$, and a production rule in F:

$$g \rightarrow h_1 ... h_m$$

such that:

$$R = R_1 h R_2, \sigma = \mathbf{mgu}(h, g)$$

and

$$Q = \sigma(R_1)\sigma(h_1)\cdots\sigma(h_m)\sigma(R_2)$$

(Here the variables of the production rule are renamed to new variables, so that all the variables in the rule do not appear in R).

Let $\Rightarrow *$ denote the reflexive and transitive closure of the relation $\Rightarrow$. As in formal language, modulo the peculiarity of the above relation $\Rightarrow$, an "derivation" is the sequence:

$$G_1 \Rightarrow^{\theta_1} G_2 \cdots \Rightarrow^{\theta_{n-1}} G_n.$$

We denote by $G_1 \Rightarrow^{\theta} *G_n$ the fact that θ is the composition of all substitutions in every direct derivation.

Definition 1.5. *The language generated by a logical grammar $GL = (I_N, I_T, X_0, F)$, is $L(GL) = \{(R, \theta) | X_0 \Rightarrow^{\theta} *R, R \in A^*_{I_T}, \theta = \theta_1 \cdots \theta_k$, k is the length of derivation for R, and θ_i is the substitution in the step $i\} \cup \{\Omega\}$*

We have some possibilities for the pair (R, θ) :

- if X_0 (or the goal G) is a ground formula, (not containing the variables) , then the substitution θ is the empty substitution, and R is $True$ or $False$, depending on the fact that G is a formula deducible or not from the set of clauses of P (by refutation);
- if X_0 contains variables, and the computation terminates, in the pairs (R, θ) we have $R \in I^*_T$, and the number of pairs is the number of solutions. If $EDB = \phi$, then $R = \lambda$. Let us denote R in the last situation by [ ], the empty clause, like usually in logic of resolution, and let θ be the answer substitution.
- if the program P is not terminating for the goal G, then $L(GL) = \{\Omega\}$, where $\Omega \notin IDB \cup EAB$.

2. Magic-set transformation

A *general deductive database* is defined as a pair $(\mathbf{D},\mathcal{L})$, where $\mathbf{D}$ is a finite set of clauses and $\mathcal{L}$ is a first order language. It is assumed that $\mathcal{L}$ has at least two symbols, one representing a constant symbol and another one representing a predicate symbol. A *definite* (resp. *normal*) database is a deductive database $(\mathbf{D},\mathcal{L})$, where $\mathbf{D}$ contains only definite (resp. normal) clauses. A *relational database* is a deductive database $(\mathbf{D},\mathcal{L})$, where $\mathbf{D}$ contains only definite facts.

Datalog is a logic programming language designed for use as a database language [2]. It is nonprocedural, set-oriented with no order sensivity, no special predicates, and no function symbols.

Consider a *Datalog* program P. As we mentioned in section 1, an *IDB predicate* is a predicate that appears in the head of some rule; if a predicate does not appear in any head, then it is an *EDB predicate*. The *EDB* (*extensional database*) is a set of relations for the *EDB* predicates; each relation is a set of tuples (or ground facts). The *IDB* (*intensional database*) is a set of relations for the *IDB* predicates. The *EDB* is the input for program P, and the *IDB* is the output and it can be computed by applying the rules of P to the *EDB*.

Because *Datalog* programs operate on potentially large databases, efficient computation of them is very important. The optimization methods are classified according to various criteria like: *formalism*, the *search strategy*, the *objective of the optimization*. There are two altenative formalisms: *algebric* and *logic*. In the case of logic formalism the evaluation of a *Datalog* goal requires building a proof tree. This tree can be constructed in two different ways: *bottom-up*, starting from the existing facts and inferring new facts, thus going towards the conclusions, or *top-down* trying to verify the premises which are needed in order for the conclusion to hold. From the objective of optimization method point of view some methods perform *program transformation*, namely, they transform a program into another program which is written in the same formalism, but which yields a more efficient computation when one applies an evaluation method to it; we refer to these as *rewriting methods*. These methods contrast with the *pure evaluation methods*, which propose effective evaluation strategies, where the optimization is performed during the evaluation itself.

The *magic sets* method is a bottom-up query evaluation technique which solves a query with particular *adornments* to the goal predicate. The program is transformed to an equivalent program using its adorned rules and *sideways information passing*. The transformed program models the constant propagation strategy of top-down methods through its *magic subgoals* added to the body of rules in the original program.

Let P be a *Datalog* program, and consider a goal on P. We can view the goal itself as defined by a rule and we add the goal rule to the program.

Definition 2.1. *An adornment of an atom* A *is an assignment either bound or free (abbreviated to* b *or* f *respectively) to each argument of* A*. An adornment of an atom with* n *arguments is denoted as a* n*-tuple. An atom* $p(t_1, ..., t_n)$ *with adornments* $< a_1, ..., a_n >$ *is denoted as* $p^{a_1,...,a_n}(t_1, ..., t_n)$ *where each* a_i *is assigned to* t_i *and is either* b *or* f.

Definition 2.2. *Given an adornment of the head of rule* r*, an argument of a subgoal of* r *is said to be* distinguished *if either:*

1. *It is a constant, or*
2. *It is a variable occuring in the head of* r *and the corresponding adornment is* b*, or*
3. *It appers in a* EDB *subgoal of* r *which has a distinguished argument.*

From this definition, variables in a *EDB* predicate occurrence are either all distinguished or all not distinguished. A *EDB* predicate occurrence with all variables distinguished is a *distinguished predicate occurrence.*

For each rule r of P, and for each adornment of the head predicate of r, we generate an *adorned rule* as follows. We consider all the distinguished arguments to be bound; this will generate an adornment for all the *IDB* predicates that are in the rule. The rule obtained by replacing all these predicates with their adorned version is an *adorned rule.*

We give distinct number to different occurences of the same predicate p in the right-hand side of a rule. For predicate p, we denote its i-th occurrence by "p_i". If there is only one occurrence of a certain predicate in the right-hand side of a rule, we may omit the occurrence number in that rule.

Definition 2.3. *We say that an adorned rule is* reachable *for the goal iff either it is the adorned rule corresponding to the goal rule, with all the* LHS *predicate arguments free, or its head predicate appears, with the same adornment, in the* RHS *of a reachable rule.*

ALGORITHM MAGIC SETS

Input:
A set of adorned rules P^A, including the goal rule, all reachable from the goal.
Output:
A new set of rules P^{magic}, equivalent to P^A with respect to the goal.
Method:

$P^{magic} := P^A$;

For each adorned rule r, and **For** each occurence of an intensional predicate p in the *RHS* of r **Do**
Begin
Generate one *magic rule* in the following way:
a) Delete all other occurrences of *IDB* predicates in the *RHS*;
b) Replace the name of p in this occurrence with $magic\_r\_p^a\_i$

where a is the adornment of p in that occurrence and i is the occurrence number;

c) Delete all nondistinguished variables of this occurrence of p, thus possibly obtaining a predicate with fewer arguments;

d) Delete all nondistinguished *EDB* predicates in r;

e) Replace the name of the head predicate p' with $magic\_p'^{a'}$ where a' is the adornment of p';

f) Delete all nondistinguished variables of p';

e) Exchange the places of the head magic predicate and the body magic predicate;

Add this rule to P^{magic}.

End

For each adorned rule r in the original program **Do**
Begin
Generate a *modified* rule in the following way:
For each occurence of an intensional predicate p in the *RHS* of r **Do**
Begin
add to the *RHS* the predicate $magic\_r\_p^a\_i(X)$ where a is the adornment of the occurence of p, i is the occurence number, X is the list of distinguished arguments in this occurrence.
If p is not the head predicate
Then the magic predicate must be inserted just before that occurrence;
Else the magic predicate is to be inserted at the beginning of the rule body, before all other literals.
End
Replace r with its modified version in P^{magic};
End

For each *IDB* predicate p and **For** each adornment **Do**
Begin
Generate a *complementary* rule as follows:
For each adorned rule r, and **For** each occurrence of p in the *RHS* of r **Do**
Begin
add the rule:
$magic\_p^a(X) :- magic\_r\_p^a\_i(X)$
where i is the considered occurrence of p, a is its adornment and X is the list of its distinguished arguments.
End

Add this rule to P^{magic}

End

Endmethod

3. Application of logic grammars formalisme

Let us illustrate the simplification of the magic-set algorithm's output by an example from [2].

Example 3.1. *The program P is:*

: r_1: $anc(X,Y) : -par(X,Y).$
: r_2: $anc(X,Y) : -anc(X,Z), par(Z,Y).$

Let us consider the goal ?-anc(X,a) which is added to the program P as the rule r_0.

The reachable adorned system P^A is:

: R_0: $q^f(X) : -anc^{fb}(X,a).$
: R_1: $anc^{fb}(X,Y) : -par(X,Y).$
: R_2: $anc^{fb}(X,Y) : -anc^{fb}(X,Z), par(Z,Y).$

As result of first DO loop, the following rules are generated:

: from R_0 $:magic\_R_0\_anc^{fb}(a).$
: from R_2 $:magic\_R_2\_anc^{fb}(Z) : -magic\_anc^{fb}(Z), par(Z,Y).$

As result of second DO loop, the following rules are generated:

: from R_0 : $q^f : -magic\_R_0\_anc^{fb}(a), anc^{fb}(X,a)$
: from R_2 : $anc^{fb}(X,Y) : -magic\_R_2\_anc^{fb}(Z), anc^{fb}(X,Z), par(Z,Y)$

As result of third DO loop, the following rules are generated:

: from R_0 $:magic\_anc^{fb}(X) : -magic\_R_0\_anc^{fb}(X).$
: from R_2 $:magic\_anc^{fb}(X) : -magic\_R_2\_anc^{fb}(X).$

Finally, the following equivalent program is obtained:

1. $:magic\_R_0\_anc^{bf}(a).$
2. $:magic\_R_2\_anc^{fb}(Z) : -magic\_anc^{fb}(Z), par(Z,Y).$
3. : $q^f : -magic\_R_0\_anc^{fb}(a), anc^{fb}(X,a)$
4. : $anc^{fb}(X,Y) : -magic\_R_2\_anc^{fb}(Z), anc^{fb}(X,Z), par(Z,Y)$
5. $:magic\_anc^{fb}(X) : -magic\_R_0\_anc^{fb}(X).$
6. $:magic\_anc^{fb}(X) : -magic\_R_2\_anc^{fb}(X).$

The logical grammar GL associated with this logic program P and the goal G is the system:

$$GL = (I_N, I_T, X_0, F)$$

where I_N is the set of predicates, I_T is as in section 1, X_0 is the goal G, F is the finite set of production rules, of the form:

1. $magic\_R_0\_anc^{fb} \rightarrow \lambda.$

2. $magic\_R_2\_anc^{fb} \rightarrow magic\_anc^{fb}, par$
3. $q^f \rightarrow magic\_R_0\_anc^{fb}, anc^{fb}$
4. $anc^{fb} \rightarrow magic\_R_2\_anc^{fb}, anc^{fb}, par$
5. $magic\_anc^{fb} \rightarrow magic\_R_0\_anc^{fb}$
6. $magic\_anc^{fb} \rightarrow magic\_R_2\_anc^{fb}$

The last two rules (5 and 6) are of type "renaming" rules. Moreover, in the clauses 5 and 6, the two pairs of predicates $magic\_anc^{fb}, magic\_R_0\_anc^{fb}$ and $magic\_anc^{fb}, magic\_R_2\_anc^{fb}$ have the same arguments: X. That means that the "rewriting" relation defined in section 1 has the property: $\forall\theta$ and $\forall G_n$, $magic\_anc^{fb} \Rightarrow^{\theta} *G_n$ iff $magic\_R_0\_anc^{fb} \Rightarrow^{\theta} *G_n$. Analogously, $\forall\theta$ and $\forall G_n$, $magic\_anc^{fb} \Rightarrow^{\theta} *G_n$ iff $magic\_R_2\_anc^{fb} \Rightarrow^{\theta} *G_n$.

As in formal grammars [7], [4], this grammar can be transformed by elimination of useless predicates $magic\_R_0\_anc^{fb}$ and $magic\_R_2\_anc^{fb}$. The obtained grammar GL' has the same generative power : $L(GL) = L(GL')$. Thus, the semantics of logic programs is the same.

The logic grammar after the elimination of "renaming" rules is the following:

: $1.magic\_anc^{fb} \rightarrow \lambda.$
: $2.magic\_anc^{fb} \rightarrow magic\_anc^{fb}, par$
: $3.q^f \rightarrow anc^{fb}$
: $4.: anc^{fb} \rightarrow par$
: $5.: anc^{fb} \rightarrow magic\_anc^{fb}, anc^{fb}, par$

The corresponding logic program is identical with the program in [2].

References

[1] K.R. Apt, M.H. van Emden *Contribution to the theory of logic programming* Journal of ACM, vol.29, 1982, pp. 841–862.

[2] S. Ceri, G. Gottlob, L. Tanca, *Logic Programming and Databases*, Spriger-Verlag, 1990.

[3] P. Deransart, J. Maluszynski *A grammatical view of logic programming* MIT Press, 1993.

[4] C.J. Hogger, *Derivation of Logic Programs*, Journal of ACM,

[5] J. Minker, *Perspective in deductive databases*, J. of Logic Programming, vol.5, 1988, pp. 33–61.

[6] D. Tatar, *Logic grammars as formal languages*, Studia Universitas "Babes-Bolyai", 1994, nr.3.

[7] J.E. Hopcroft, J.D. Ullman, *Introduction to automata theory, languages and computation*, Springer Verlag, 1979.

Babeş-Bolyai University, Faculty of Mathematics and Informatics,
RO 3400 Cluj-Napoca, str. Kogălniceanu 1, Romania
E-mail address: {dtatar,ivarga}@cs.ubbcluj.ro