STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXIX, 3, 1994

LOGICAL GRAMMARS AS A TOOL FOR STUDYING LOGIC PROGRAMMING

Doina TATAR[*]

Dedicated to Professor Sever Groze on his 65th anniversary

Received: March 11, 1995
AMS subject classification: 68Q50, 68Q55, 68T27

REZUMAT. - Gramatici logice ca instrument în studiul programării logice. Articolul defineşte o nouă semantică operaţională a programelor logice, semantică bazată pe limbaje formale şi pe interpretări care pot conţine atomi cu argumente variabile. Este demonstrată corectitudinea şi completitudinea acestei semantici. În acest mod, anumite rezultate bine cunoscute în teoria limbajelor formale pot fi utilizate în studiul programelor logice.

Abstract. The paper defines a new operational semantics for logic programs, which is based on framework of formal languages and on interpretations containing possibly non-ground atoms. The soundness and completness are shown to hold. Thus, some folklore results in formal languages can be utilized for improvement of logic programs.

1. Introduction. In recent years there has been a great deal of interest in logic programming languages. The first step in this development was the seminal paper by van Emden and Kowalski [14],in which they outlined declarative and operational semantics of Horn Clause Logic (HCL) as a programming language.Apt and van Emden [1] built upon the former work and defined the fixpoint semantic of HCL.From one point of view,any attempt at formulating a semantic for a program P in a logic programming language (as PROLOG) is simply closed,as programs are statements in the HCL fragment of first-order logic. On the other hand, from a computational point of view, this semantics is not completely adequate, as they ignore several behavioural aspects of logic programs. These

[*] *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

include issues such as termination, the answer substitution, the search strategy, etc. This is the reason for which many authors attempt to define new semantics for logic programs: [5], [6], [10], [13].

In this paper we propose a new semantic for operational behaviour of logic programs, which is based on the framework of formal languages. Several direct correspondences that can be made for transforming and optimising logic programs are pointed out. In non-monotonic logic programming this framework is very useful also. We think that an deeper connexion between these two fields, apparently so different, can be to benefit by logic programming:the formal languages(tree-languages) and the rewriting systems are very intensively studied since many years.

2. Preliminaries.

The language considered here is essentially that of first-order predicate logic without function symbols. Let:

- P 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

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

where $p \in P$ with arity 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 $P' \subseteq P$ let $A_{P'}$ be the set of atoms with predicate symbols from P'. In some recent papers [7], [9], [13], the set of predicates is considered as divided into two disjoint sets:the set EDB of extensional predicates (or extensional databases predicates) which represent basic "facts", and the set IDB(of intensional databases predicates) representing facts deduced from the basic facts via the logic program. Particularly, the set EDB can be the empty set (as, for simplicity, in most of the following demonstrations).

Definition 2.1 A logic program P is a sequence of Horn clauses,
that is, clauses of the form:

$$p \leftarrow q_1,\dots,q_n$$

where p and $q_1,\dots,q_n$ 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 $(q_1,\dots,q_n)$ of a clause as its head and body. A clause is logically interpreted as the universal closure of the implication $q_1 \wedge\dots\wedge q_n \rightarrow p$.
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 2.2 A goal G consists of a conjunction of atoms, such that a successfully terminating computation corresponds to a demonstration of this goal by refutation (SLD-refutation), and is denoted by:

$$\leftarrow r_1,\dots,r_t$$

3. Logical grammars

Over time, we may want to "apply" the same IDB for many quite different EDB's. In this context, and because IDB is the "core" of logic program, the properties of the IDB merit careful study. Recent papers have addressed the problems of studying and optimizing logic programs from various points of view. ([2], [3], [4], [7]) Our tool is the logical grammars:

Definition 3.1 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 rule, of the form:

$$a)\ p \rightarrow q_1 \ldots q_m,\ m \geq 1$$

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

or

$$b)\ p \rightarrow \lambda$$

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 $\sigma = mgu(g,h)$ of atoms g and h are defined as in logic programming. ([1], [2], [3], [12]).

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

Definition (3.2) If $R \in A^+$ and $Q \in A^*$, then:

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

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

$$g \rightarrow h_1 \ldots h_m$$

such that:

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

and

$$Q = \sigma(R_1)\, \sigma(h_1) \ldots \sigma(h_m) \sigma(R_2)$$

(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 $\Rightarrow$. If θ is the composition of all substitutions in every direct derivation,let denote it by $\Rightarrow^{\theta\,*}$.

Definition (3.3) For a logical grammar $GL = (I_N, I_T, X_0, F)$ the generated language is L(GL), where:

L(GL) = $\{(R,\theta) \mid X_0 \Rightarrow^{\theta} * R, R \in A^{*}_{I_T}$, $\theta = \theta_1 \ldots \theta_k$ 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 P (by refutation).
- if X_0 contains variables, and the computation is succesfully terminating, in the pairs (R,θ) we have $R \in I^{*}_{T}$, and the number of pairs represents the number of solutions. If EDB= ϕ, then $R = \lambda$. Let denote the last situation by R=[], the empty clause, like usually in logic,and let θ the answer substitution.
- if the program P is not terminating for the goal G, then L(GL) =$\{\Omega\}$, where $\Omega \notin$ IDB $\cup$ EAB.

Example

Let P be the program:

```
domains
        lista = symbol*
predicates
        m(symbol, integer, lista)
        consec(symbol, symbol, lista)
clauses
        consec(U,V,X):-m(U,I,X),m(V,I+1,X).
        m(U,1,U.X).
        m(U,I+1, V.X):-m(U,I,X).
```

If the goal is: G=consec (c,X,a.b.c.d.nil)

then the derivation is:

consec(c,X,a.b.c.d.nil) $\Rightarrow^{\theta 1=\lambda}$ m(c,I,a.b.c.d.nil)m(X,I+1,a.b.c.d.nil)$\Rightarrow^{\theta 2=)I/I'+1)}$ m(c,I', b.c.d.nil) m(X,I'+2,a.b.c.d.nil)$\Rightarrow^{\theta 3=(I'/I''+1)}$ m(C,I",c.d.nil)m(X,I"+3,a.b.c.d.nil)$\Rightarrow^{\theta 4=(I''/1)}$ m(X,4,a.b.c.d.nil)$\Rightarrow^{\theta 5=\lambda}$ m(X,3,b.c. d.nil)$\Rightarrow^{\theta 6=\lambda}$ m(X,2,c.d.nil) $\Rightarrow^{\theta 7=\lambda}$ m(X,1,d.nil)$\Rightarrow^{\theta 8=(X/d)}\lambda$.

The substitution in variable X only is:

$$\theta = \theta_1\theta_2...\theta_8 = (X/d)$$

and the pair $([],\theta) \in L(GL)$.

4. Remarks about soudness and completness of logical grammars

For the purpose of simpler manipulation , let us denote
the logical grammar GL with the symbol initial X_0 by GL_{X_0} and the pairs in the logical language $L(GL_{X_0})$ by the triplets (P,θ, X_0). Let us, furthermore suppose $EDB=\phi$. Then, the definition (3.3) becomes :

$$L(GL_{X_0}) = \{([],\theta,X_0)|X_0 \Rightarrow^{\theta} * []\} \cup \{\Omega\}$$

In [6] the "success set " for a logic program p is defined as:
succ(P) = $\{G(t_1,...,t_n|t_1,...,t_n$ are terms in a standard Herbrand interpretation, the.. they are grounded terms, and the goal $G(t_1,...,t_n)$ is deductible from P}
or, equivalently,
succ(P) = $\{G(t_1,...,t_n)|t_1,...,t_n$ are terms in a standard Herbrand interpretation and there exists a SLD-refutation of $G(t_1,...,t_n)$ from P}
The set succ(P) is not completely adequate as operational semantics since it hides one fundamental aspect of logic programming: the ability to compute substitutions. A more adequate definition should be the following:
succ'(P) = $\{(G(t_1,...,t_n),\theta)|t_1,...,t_n$ are non-ground terms, and $G(t_1,...,t_n)$ has a SLD-refutation with computed answer $\theta\}$.

Let us remark that in [6] two other models (S-model and C-model) are introduced by permitting that $t_1,...,t_n$ are not necessarily ground terms.

In the sequel we will see the connexion between the set succ'(P)and the logical grammars introduced in paragraph 3.

Definition 4.1 Let U'_H be a nonstandard Herbrand interpretation (which admits non-ground terms). We denote by L_p and we will call them the total language of a logical program P, the following language:

$$L_p = \bigcup_{X_0 \in U'_N} L(GL_{X_0})$$

Conformally with the previous definitions,

L_p={([],θ,G)|G is an arbitrary goal, G $\Rightarrow^{\theta}$ * []}∪{(Ω,G)|P is cycling for the goal G} ∪ {(False,G)|G has not a SLD-refutation from P}. Let prove that this language L_p represents a sound and complete semantic for P.

Lema 4.2

If G$\Rightarrow^{\sigma}$ G' and (G',θ) ∈ succ'(P) then (G,σθ) ∈ succ'(P).

Demonstration

We denote shortly for (G',θ) ∈ succ'(P) by P → G'θ, where " → " is logical implication and G'θ is the conjunction of atoms of the goal G'θ.

Suppose G = $A_1,\dots,A_s,\dots,A_k$ and

G'= $A_1\sigma,\dots,A_{s-1}\sigma,B_1\sigma,\dots B_m\sigma, A_{s+1}\sigma,\dots,A_k$ such that the rule

A →$B_1\dots B_m$ was applied in the rewriting G ⇒ G', with σ =mgu(A_s,A).

By logical means of applied rule results that $B_1 \wedge\dots\wedge B_m$ → A

and by syllogism we have sequently :

P → G'θ, and G'θ → $A_1\sigma\theta\wedge\dots\wedge A_{s-1}\sigma\theta\wedge A_s\sigma\theta\wedge\dots\wedge A_k\sigma\theta$

Results that P → Gσθ, such that (G,σθ) ∈ succ'(P).

Theorem 4.3 (of soundness)

Let P be a logic program and G a goal. If ([],θ,G) ∈ L_p then (G,θ)∈ succ'(P).

Demonstration Let G = $A_1,\dots,A_k$ and assume that ([],θ,G) ∈ L_p

We prove by induction on the length n of deduction of ([],θ,G) that (G,θ)∈ succ'(P). If n=1 then exists a unit clause A_1 in P such that $A_1 \Rightarrow^{\theta_1}$[].

Then (θ_1,A_1) ∈ succ'(P).

Let G $\Rightarrow^{\theta}$ * [] be a deduction of length n. Then it can be detailed as:

$$G \Rightarrow^{\theta_1} G_1 \Rightarrow^{\theta_1} \dots \Rightarrow^{\theta_n} []$$

or

$$G \Rightarrow^{\theta_1} G_1 \Rightarrow^{\theta'} * []$$

where $G_1 \Rightarrow^{\theta'} * []$, $\theta' = \theta_2 \ldots \theta_n$ is a deduction of length n-1.
Suppose that $G = A_1, \ldots, A_s, \ldots, A_k$
and $G_1 = A_1\sigma, \ldots, A_{s-1}\sigma, B_1\sigma, \ldots, B_m\sigma, A_{s+1}\sigma, \ldots, A_k$
where $\sigma = mgu(A_s, A)$, $A \leftarrow B_1, \ldots, B_m$ is a clause in P. (That means
$G \Rightarrow^{\sigma} G_1$, $\theta_1 = \sigma$).
By induction hypothesis and, on the other hand, from $G \Rightarrow^{\theta_1} G_1$ and lema (4.2) we obtain
$(G, \theta_1, \theta') = (G, \theta) \in$ succ'(P).

Theorem 4.4 (of completeness)

Let P be a logic program and G a goal. If $(G,\theta) \in$ succ'(P), then
$([], \theta, G) \in L_p$.

Demonstration By induction on the length of a logical implication of $G\theta$ from P.

5. Applications

For a logic program P let denote by GL_p the associated logic grammar like in definition (3.3).
We can define two kinds of equivalence of the logic programs:

Definition (5.1) Two logic programs P_1 and P_2 are strong equivalent if $\forall X_0$ we have
$L(GL_{P_1}) = L(GL_{P_2})$ (or, in the formal languages terminology, if GL_{P_1} and
GL_{P_2} are equivalent for every goal X_0).
The equivalence of two programs in the following is the strong equivalence, therefore the equivalence for same goals.

Definition 5.2 Two logic programs P_1 and P_2 are equivalent if $L_{P_1} = L_{P_2}$
where L_p is defined like in definition 4.1.
The consequence of the introduced notion is the possibility of the definition for some transformations about logic programs, such that the obtained programs are equivalent with the initial programs.

Definition 5.3 [11]. A logic program P is said to be "constant-free", if the atom in left side of each clause contains no constants.

Theorem 5.4 Every logic program P can be transformed into an equivalent "constant-free" program P', if the below procedure C is not failing.

Proof: Let observe that a program P is "constant-free" if associated logic grammar GL_p is such that in every production rule, the atom in left side (a nonterminal atom) contains no constants. We will apply the following procedure C, while remains a production rule with an atom containing constants in his left side.

Procedure C: Let $a = x_j$ be a constant contained in the nonterminal atom $p(x_1,...x_n)$ in one of his occurs in left side of a production rule.

Rule C1. Replay every production rule of the form:

$$p(y_1,...,y_n) \rightarrow h_1...h_k$$

such that $y_j = a$ by a new production rule

$$p_{j,a}(y_1,...,y_{j-1}\ ,\ y_{j+1},...,y_n) \rightarrow \sigma(h_1)...\sigma(h_k)$$

where $\sigma = [(y_j/a)]$. or $\sigma = \text{mgu}\ (\ y_j, a\ )$

Let observe that we can have, in another rule with nonterminal p in left side, the case $y_j = b$ and b is not a. Rule C1 introduces in this case the new nonterminal $p_{j,b}$ $(y_1,...,y_{j-1},y_{j+1},...y_k$. Then, the rule C1 cannot introduce a failing situation for the procedure C.

Rule C2. Replay every production rule of the form:

$$q \rightarrow l_1...p(y_1,...,y_n)...l_m$$

by a new production rule:

$$\sigma(q) \rightarrow \sigma(l_1)...\sigma(p_{j,a}(y_1,...,y_{j-1},y_{j+1},...,y_n)...\sigma(l_m)$$

where $\sigma = \text{mgu}(a,y_j)$, if σ does exists. Unlike rule C1, if yj=b and b is not a, then σ does not exist and procedure C fails. Theorem 5.4 results by induction on the length of a successful derivation. It is enough to prove that, if:

$$Q,R \in A^*{}_{I\mu I_N}, Q \Rightarrow * GL_p R$$

then

$$Q' \Rightarrow GL_{p'} * R'$$

where P' is the logic program obtained by successfully application of procedure C ,and Q',R' are obtained from Q and R by replacing a non constant-free predicates p by their corresponding predicate ($p_{j,a}$).
If the length is 1,then $Q' \Rightarrow * GL_{p'}R'$ is true, by once application of rule C_1 or C_2.
Suppose that, for every derivation of length n, we have implication verified. Let $Q \Rightarrow^*_{GL_p} R$ be a derivation of length n+1, and let point out the last step :

$$Q \Rightarrow^*_{GL_p} T \Rightarrow R$$

By induction hypothesis, we have $Q' \Rightarrow^*_{GL_{p'}} T'$. If $T \Rightarrow_{GL_p} R$ by utilising a rule $ C_1 containing the non constant -free predicate p in left hand-side ,then $T = T_1pT_2$ and $R = T_1h_1...h_kT_2$. By induction hypothesis $T' = T'_1p_{j,a}T'_2$, where T'_1 and T'_2 are corresponding to T_1 and T_2. Then:

$$T'_1p_{j,a}T'_2 \Rightarrow T'_1\sigma(h_1)...\sigma(h_k)T'_2$$

and $R' = T'_1\sigma(h_1)...\sigma(h_k)T'_2$ is corresponding to R .
If $T \Rightarrow_{GL_p} R$ by utilizing a rule of the form C_2, then $T = ._1qT_2$, $R=T_1l_1...p(y_1,...y_n)...l_mT_2$. By induction hypothesis $T' = T'_1\sigma(q)T'_2$ and thus $R' = T'_1\sigma(l_1)...\sigma(p_{j,a})(...))...\sigma(l_m)T'_2$ is corresponding to R.q.e.d.

Another syntactical improvement of a logical program is the elimination of the predicates that have the equal arguments.

Definition 5.5 [11] An atom is "loop-free" if all the arguments are different. A production rule is "loop-free" if the nonterminal atom in the left hand side of this production rule is "loop-free". A logic grammar is "loop-free" if all his production rules are "loop-free". A logic program P is "loop-free" if GL_p is "loop-free". Fortunately, a similar with (5.4) theorem can be proved by induction:

Theorem 5.6 Every logic program P can be transformed into an equivalent "loop-free" program P'.

Also, in a logic grammar, as well as in context-free grammars, we can remove useless predicates, without affecting the generated language.

Definition 5.7 [12] A nonterminal atom p is productive if exists a derivation

$$p \Rightarrow *Q,\ Q \in A^{*}{}_{I_r}$$

A nonterminal atom p is deductible if exists a derivation:

$$X_0 \Rightarrow * \ P,\ P \in A^{*}{}_{I_N \cup I_r}$$

such that P contains p.

A logic grammar GL is reduced if every nonterminat atom is deductible and productive. A logic program P is reduced if the associated logic grammar GL is reduced.

Theorem 5.8 Every logic program P can be transformed into an equivalent reduced program P'.

Proof: By application of the algorithms for obtaining a reduced context-free grammar, to the logic grammar GL_P.

REFERENCES

1. K.R. Apt,M.H.van Emden : "Contribution to the theory of logic programming " J. of ACM, vol.29,1982, pg.841-862.
2. K.R. Apt, D. Pedreschi: Studies in pure Prolog:termination, CWI Report CS-R9048, September, 1990.
3. K.R. Apt, D. Pedreschi: Proving termination of general Prolog programs, CWI Report CS-R9111, February, 1991.
4. S.Debray, D.Waren: "Functional Computation in Logic Programs", ACM Transaction, vol.11, 1989, pg.451-481.
5. S.Debray,P.Mishra:"Denotational and operational semantics for PROLOG", The Journal of Logic Programming, vol.5, nr.1, 1988, pp.33-61.
6. M.Falaschi, G.Levi, M.Martelli, G.Palamidessi: "Declarative modelling of the operational behaviour of logic languages" report Univ.di Pisa,TR-10/1980.I. Guessarian: Some fixpoint techniques in algebraic structures
7. H. Gaifman, H.Mairson,Y.Sagiv,M.Y.Vardi:" Undecidable Optimization Problems for Database Logic Programs", Journal of ACM, July, 1993, pp. 683-714.
8. C.J. Hogger: "Derivation of Logic Programs", Journal of ACM, April, 1981, pp. 372-393.

9. J.Minker:"Perspective in deductive databases" J.of Logic Programming, vol.5, 1988, pp.33-61.
10. T.Przymusinski:"On the declarative and procedural semantics of logic programs" J. of Automated Reasoning, vol5, 1989, pp.167-205.
11. I. Shioya: "Logic hypergraph grammars and context-free hypergraph grammars", Systems and Computers, vol.22, nr.7, 1991.
12. D.Tatar:" Utilizarea gramaticii sintactice asociata unei scheme de recursie, în studiul programelor", Studii si cercetari matematice, nr.4,
1988, pp. 337-347.
13. A. van Gelder, K.A.Ross, J.S.Schlipf.:"The Well-Founded Semantics for General Logic Programs", Journal of ACM, July, 1991, pp. 620-651.
14. M.H.van Emden, R.A.Kowalski;"The semantics of predicate logic", J. of ACM, oct. 1976, pp. 733-742.