THE DENOTATIONAL SEMANTICS OF HORN CLAUSES AS A PRODUCTION SYSTEM

J-L. Lassez and M. Maher

Dept. of Computer Science
University of Melbourne
Parkville, Victoria, 3052
Australia.

ABSTRACT

We show how one of Nilsson's tenets on rule-based production systems, when applied to Horn clause programs, leads to a denotational semantics. This formalism, in turn provides a striking illustration of a second Nilsson tenet.

I PRELIMINARIES

The three properties of a denotational semantics (McGettrick, 1980; Tennent 1981) that we consider here are the following:

1. It is a functional semantics, that is the meaning of a segment of program S is a function denoted $[\![S]\!]$ over a set of states.

2. The definitions of these semantic functions are structured in such a way that the meaning of any composite phrase is expressed in terms of the meanings of its immediate constituents. For example, in many conventional programming languages $[\![S_1;S_2]\!] = [\![S_2]\!] \circ [\![S_1]\!]$.

3. The function assigned to a recursive definition is defined as the least fixedpoint of a suitable operator.

We will consider here two of Nilsson's tenets from his "Principles of Artificial Intelligence" (Nilsson, 1982). The first tenet is fundamental for our purpose, allowing us to define a Horn clause program as consisting only of a set of rules, the facts becoming the input.

> "Wffs representing assertional knowledge about the problem are separated into two categories: rules and facts. The rules consist of those assertions given in implicational form. Typically they express general knowledge about a particular subject area and are used as production rules. The facts are the assertions that are not expressed as implications. Typically they represent specific knowledge relevant to a particular case."

* This research supported by the Australian Computer Research Board

The second tenet links commutativity and decomposition:

> "Under certain conditions the order in which a set of applicable rules is applied to a database is unimportant. When these conditions are satisfied, a production system improves its efficiency by avoiding needless expansion of redundant solution paths." Such conditions may be commutativity or decomposability. Furthermore, Nilsson shows that in some cases there are relationships between commutative production systems and decomposable ones.

The concept of decomposition that will be used here is the splitting of the set of rules (instead of Nilsson's splitting of the initial database), in such a way that the two resulting programs can be run separately.

II DENOTATIONAL SEMANTICS OF HORN CLAUSES

Horn clause programs, which form the theoretical basis for PROLOG programs (Kowalski, 1979), can be defined as a finite set of definite clauses.

A definite clause is of the form

$$A \leftarrow B_1,\ldots,B_n$$

where A, B_1, ..., B_n are atomic formulae and where $n \geq 0$.

Example:

```
path(A,B) ←
path(B,C) ←
path(D,E) ←
path(x,y) ← path(y,x)
path(x,z) ← path(x,y), path(y,z)
```

The usual semantics (van Emden and Kowalski, 1976) of this program is given as a set: the set of all paths in the graph which is defined by the first three clauses. However it is clear that the nature of these three clauses differs from that of the remaining two - the latter express general properties (symmetry and transitivity) whereas the former specify one particular graph. Hence it is

natural to separate the two types of clauses: the rules (those clauses in implicational form) and the others which represent the initial database of knowledge. In the remainder of this paper we consider a program to be the conjunction of rules in a set R, which takes as input an initial database.

To a set R of rules we associate a function $[\![R]\!]$ which maps a set of facts into the set of facts which can be immediately deduced from the first set by a single application of a rule in R. Thus

$A \in [\![R]\!](I)$ iff
there exists a clause $B_0 \leftarrow B_1,\ldots,B_n$ in R

and a substitution θ such that $A = B_0\theta$ and $\{B_1\theta,\ldots,B_n\theta\} \subseteq I$.

Informally the semantics of the program P is the function $[\![P]\!]$ which maps an initial database I into the set of all facts which can be deduced from this database by repeated application of the rules in R. So we have

$$[\![P]\!](I) = \bigcup_{k=0}^{\infty} ([\![R]\!]+Id)^k(I) \qquad (*)$$

where Id is the identity function and $(f+g)(X) = f(X) \cup g(X)$. For simplicity of expression we will write this as

$$[\![P]\!](I) = ([\![R]\!]+Id)^{\omega}(I)$$

Equivalently, $[\![P]\!]$ can be defined as the least fixedpoint of the functional τ by the following proposition (Lassez and Maher, 1983).

Proposition:

$[\![P]\!]$ is the least fixedpoint of the operator τ where $\tau(f) = [\![R]\!] \circ f + Id$

It is clear that properties 1 and 3 of denotational semantics are satisfied by this definition. The second property, which gives the semantics of the whole program in terms of the semantics of its components (in this case rules), is given by the following theorem. This theorem, which is a variant of a theorem of (Tarski, 1955), is proved in (Lassez and Maher, 1983). Let $R_1,\ldots,R_n$ be the individual rules of R with the corresponding programs $P_1,\ldots,P_n$.

Theorem 1:

$$[\![P]\!] = [\![R_1 \wedge \ldots \wedge R_n]\!] = ([\![P_1]\!] \circ \ldots \circ [\![P_n]\!])^{\omega}$$
$$= (([\![R_1]\!]+Id)^{\omega} \circ \ldots \circ ([\![R_n]\!]+Id)^{\omega})^{\omega}$$

It can be shown (Lassez and Maher, 1983) that these definitions and results are compatible with the semantics of (van Emden and Kowalski, 1976). If the R_i's represent sets of rules, rather than individual rules, then this theorem is still valid, provided every rule in R is contained in some R_i.

In (*) and in (van Emden and Kowalski, 1976) the rules are applied in parallel to the database. This parallelism is not required by the informal semantics. Theorem 1 shows that we can generate the same set of facts by applying the rules in a completely different way. In fact Theorem 1 can be generalized to show that the order of application of the rules is irrelevant, provided a condition of fairness is met, which corresponds more closely to the informal semantics (Lassez and Maher, 1983).

III COMMUTATIVITY AND DECOMPOSITION

The following theorem establishes that if the order of application of the two (not necessarily disjoint) sets of rules R_1 and R_2 does not matter - they can even be applied in parallel - then the program P can be divided into two subprograms P_1 and P_2 such that $[\![P]\!] = [\![P_1]\!] + [\![P_2]\!]$. In this case SLD resolution (Apt and van Emden, 1982) can be used in parallel for P_1 and P_2, the two search spaces involved being in general far smaller than the whole search space for P, the construction of which necessitates combinations of rules from both programs.

Theorem 2:

If $([\![R_1]\!]+Id) \circ ([\![R_2]\!]+Id) = ([\![R_2]\!]+Id) \circ ([\![R_1]\!]+Id)$
$= [\![R]\!] + Id$

then $[\![P]\!] = [\![P_1]\!] + [\![P_2]\!]$

If we consider closures of rules (i.e. $[\![P_1]\!]$ and $[\![P_2]\!]$ instead of $[\![R_1]\!]$ and $[\![R_2]\!]$) then we have a condition which is both necessary and sufficient for P to be decomposable.

Theorem 3:

$[\![P]\!] = [\![P_1]\!] + [\![P_2]\!]$ iff

$[\![P_1]\!] \circ [\![P_2]\!] = [\![P_2]\!] \circ [\![P_1]\!] = [\![P_1]\!] + [\![P_2]\!]$

These theorems are related to classical results of Ore on closure operators. The proofs

can be found in (Lassez and Maher, 1983).

Examples:

Consider R = { N(x) ← N(S(x)),
N(S(x)) ← N(x) }

where the Herbrand Base HB is { N($S^n(0)$) : n=0,1,... } We call S the successor function. One verifies easily that ($\mathbb{[}R_1\mathbb{]}$+Id) o ($\mathbb{[}R_2\mathbb{]}$+Id) = ($\mathbb{[}R_2\mathbb{]}$+Id) o ($\mathbb{[}R_1\mathbb{]}$+Id) = $\mathbb{[}R\mathbb{]}$+Id. Therefore one can break the connection between the two rules and perform SLD resolution on each of them separately. That is for a given A ∈ HB we look simultaneously in parallel for successors only and for predecessors only. Without this split, SLD resolution would search alternatively for predecessors and successors creating a search space "exponentially" larger than the two preceding ones.

The hypotheses of the theorems do not require that R_1 and R_2 be disjoint sets of rules as we show in the next example. A database on military personnel is processed by the following program P. The data is of the type: C(X,Y) (that is X is in the same company as Y), Att(X,B052) (X sleeps in barrack 052), Att(X,C.SMITH) (X's commanding officer is Captain Smith), ... Each attribute is characteristic of a company and associated to all its members.

C(x,y) ← C(y,x)
C(x,z) ← C(x,y), C(y,z)
C(x,y) ← Att(x,c), Att(y,c)
Att(x,c) ← C(x,y), Att(y,c)

This can be divided into two separate programs which, this time, have overlapping sets of rules. P_1 is

C(x,y) ← C(y,x)
C(x,z) ← C(x,y), C(y,z)
Att(x,c) ← C(x,y), Att(y,c)

and P_2 is

C(x,y) ← C(y,x)
C(x,y) ← Att(x,c), Att(y,c)
Att(x,c) ← C(x,y), Att(y,c)

It can be verified that $\mathbb{[}P\mathbb{]} = \mathbb{[}P_1\mathbb{]} + \mathbb{[}P_2\mathbb{]}$.

We now have two simpler programs which can be executed in parallel. That portion of the search space caused by the the unnecessary interactions between the second and third rules of P (via the remainder of the program) has been discarded.

REFERENCES

[1] Apt, K. R. and M. H. van Emden, "Contributions to the theory of logic programming" J.ACM 29:3 (1982) 841-862.

[2] van Emden, M. H. and R. A. Kowalski, "The semantics of predicate logic as a programming language" J.ACM 23:4 (1976) 733-742.

[3] Kowalski, R. A. Logic for Problem Solving North-Holland, 1979.

[4] Lassez, J-L. and M. Maher, "Closures and fairness in the semantics of programming logic" Theor. Comput. Sci. (to appear). Also Technical Report 83/3, Dept. of Computer Science, University of Melbourne, 1983.

[5] McGettrick, A. D. The Definition of Programming Languages. Cambridge University Press, 1980.

[6] Nilsson, N. J. Principles of Artificial Intelligence. Springer Verlag, 1982.

[7] Tarski, A. "A lattice-theoretical fixpoint theorem and its applications" Pacific J. Math. 5 (1955), 285-309.

[8] Tennent, R. D. Principles of Programming Languages. Prentice-Hall, 1981.