Maintaining Consistency in a Stratified Production System Program

Louiqa Raschid
Department of Information Systems and
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742
louiqa@secd.cs.umd.edu

Abstract

We present our research on defining a correct semantics for forward chaining production systems (PS) programs. A correct semantics ensures that the execution of the program will not produce incorrect answers and execution will terminate; it also ensures that the answers are consistent. We define a class of *stratified* PS programs, and propose an operational semantics for these programs. We define an operator T_{PS}, which computes the operational fixpoint for the productions of the stratified PS program; the fixpoint captures the meaning of the PS program. The theory that can be derived from the productions of the PS program may be inconsistent with the constraints that are also derived from the PS program. We can then view the constraints as modifying the theory so that the modified theory PS is consistent with the constraints. However, the same answers are obtained in the operational semantics of the stratified PS program or from the modified theory PS.

1. Introduction

In recent years, much AI research and development has focused on forward chaining rule–based systems which follow the *production sytem* (PS) paradigm (Hayes–Roth 1985). Large production rule–based expert systems have been successfully developed in diverse domains such as engineering design databases, trouble–shooting in telephone networks, and configuring VAX computer systems. In these domains, the expert system programs often have to reason with large quantities of data. As the production rule base and the database grow larger, these programs have to access information stored on disk. Thus, for performance reasons, it is important that PS programs be implemented using database technology. Research in this area is reported in Delcambre and Etheredge 1988, Maindreville and Simon 1988, Raschid, Sellis and Lin 1988, Simon and Mandreville 1988, Sellis, Lin and Raschid 1988 and Widom and Finkelstein 1989.

If large production systems (PS) are to be implemented successfully to interface with large (relational) databases, then it is critical that the semantics of PS be well understood. Unfortunately, most PS have an incomplete operational semantics defined for them. This can result in non–terminating execution of productions and inconsistent answers.

In this paper, we describe our research on defining a *correct* semantics for PS programs. This paper is organized as follows: In section 2, we introduce the operational semantics of OPS5, an example of a PS (Forgy 1981 and Forgy 1982) and motivate this research using some example OPS5 programs. We also define how a logical theory can be derived from the productions of the PS program. In section 3, we define a class of *stratified* PS programs and define an operational semantics for stratified PS programs. A stratified PS program is a stratified program, and comprises an extensional database (EDB) of facts, a stratified intensional database (IDB) of rules, and a stratified set of integrity constraints (IC). Both rules and integrity constraints correspond to productions in the PS program. We show that processing is guaranteed to terminate upon reaching the operational fixpoint of a defined operator T_{PS}. The fixpoint captures the meaning of the PS program and is correct and consistent.

This research was sponsored partially by the National Science Foundation under Grant DMC8814989 and by the University of Maryland Institute for Advanced Computer Studies.

The author would like to thank Timos Sellis, Anne Litcher and Arcot Rajasekar.

The theory of the PS program, comprising the EDB facts and the IDB rules that are derived from the productions, may be inconsistent with the integrity constraints (IC) that are also derived from the productions. We can view the IC as modifying the theory so that the modified stratified theory *PS* will be consistent with IC. This is described in section 4. We also show the equivalence between the answers obtained in the operational fixpoint and the minimal model for the theory *PS*.

2. The OPS5 Production System

In this section we introduce the operational semantics defined for the OPS5 production system language and highlight some of its problems. We chose the OPS5 production system language (Forgy 1981 and Forgy 1982) as an example.

Operational Semantics of OPS5

An OPS5 knowledge base comprises a set of productions and an extensional database (EDB) of ground positive unit clauses which may be stored in relations. There will be one relation corresponding to each predicate.

An OPS5 production consists of (1) the symbol **p**, (2) the **name** of the production, (3) the **antecedent** or the left hand side (LHS), (4) the symbol →, and (5) the **consequent actions** or the right hand side (RHS), enclosed within parentheses.

The **antecedent** is a conjunction of first order positive literals of the form:
$\forall\ x_1, x_2,.. x_n\ P(a_1, a_2,..a_m, x_1, x_2,..x_n)$
or negative literals of the form:
$\forall\ x_1, x_2,.. x_n\ \neg Q(a_1, a_2,..a_m, x_1, x_2,..x_n)$. P and Q are (n+m)–ary predicates corresponding to the EDB relations, $a_1, a_2,...a_m$ are constants and $x_1, x_2,..x_n$ are variables. Assume that all variables are *range–restricted*, i.e., any variable that occurs in a literal must appear in a positive literal. The advantages of this restriction have been discussed in Sadri and Kowalski 1988 and correspond to the safety of evaluating queries.

The antecedent of each production is interpreted as a query against the EDB relations. For example, for each of the positive literals, $P(a_1, a_2,..a_m, x_1, x_2,..x_n)$, relation P is queried, and a set of *instantiated* tuples of P satisfying each positive literal in the antecedent is retrieved. For each of the negative literals, Q, the query corresponding to the first order formula, $\neg\ (\ \exists\ x_1, x_2,..x_n\ Q(a_1, a_2,..,a_m, x_1, x_2,..,x_n)\ )$, is verified against the relation Q. Note that since the variables are range–restricted, the queries corresponding to the negative literals can always be verified. The antecedent of a production is *satisfied* if the relations contain instantiated tuples corresponding to each of the positive literals and the relations do not contain tuples corresponding to the negative literals.

The *consequent actions* are of the form: (**make** $R(a_1, a_2,..,a_m,x_1, x_2,..,x_n)$) or (**remove** $P(a_1, a_2,..,a_m,x_1, x_2,..,x_n)$), where P and R are relations. Again, we assume that all variables $x_1, x_2, \ldots, x_n$ are range restricted. The interpretation of the **make** action is to insert the corresponding tuple into the R relation. Similarly, the **remove** action deletes the existing tuple from the P relation. This requires that the expression $P(a_1, a_2,..,a_m,x_1, x_2,..,x_n)$ that is referred to by the **remove** action must occur as a positive literal in the antecedent of that production.

We note that the OPS5 language has many additional features that are not described here; we have not considered the effect of such features in our research.

The operational semantics of an OPS5 program has been defined as follows: The initial state of the PS corresponds to an initial EDB, corresponding to all the tuples of the EDB relations, and a set of productions. Processing in a production system repeatedly cycles through the following sequence:

Match
For each production, the antecedent, interpreted as above, is queried against the tuples of the corresponding relations. Each production whose antecedent is satisfied, together with its *instantiated* tuples is placed in a *conflict set.*

Select
Select one satisfied production from the conflict set. If there is no such production, halt execution.

Act
For this selected production, execute the **make** or **remove** action, interpreted as above. As a result of the **Act** phase, the EDB relations are updated. Consequently, the next **Match** phase may produce a new conflict set.

There is no concept of a query that retrieves information, in a PS program. Processing will continue as long as productions are executed and the EDB relations are updated. Processing terminates when an operational fixpoint is reached, i.e., when there are no longer any productions that can update the EDB relations. This operational

fixpoint (or the updated relations) corresponds to the meaning of the PS program.

Shortcomings of the Operational Semantics

The operational semantics for OPS5 are incompletely defined. Consequently, an initial EDB of relations and a set of productions can produce different answers. In some cases, an operational fixpoint is never reached.

Example 1

Consider a PS whose initial EDB has two tuples, {Employee(Mike). GoodWorker(Mike).}, and the following set of productions:

(p p_1 (Employee(X), GoodWorker(X)) →
(**make** Manager(X)))
(p p_2 (Manager(X)) → (**make** HasOffice(X)))
(p p_3 (Employee(X), HasOffice(X)) →
(**make** PoorWorker(X)))
(p p_4 (Manager(X), PoorWorker(X)) →
(**remove** Manager(X)))

Given this initial EDB and corresponding set of productions, p_1, p_2 and p_3 will execute in that sequence and the tuples Manager(Mike), HasOffice(Mike) and PoorWorker(Mike) will be added to the corresponding EDB relations. Next, p_4 executes and the tuple Manager(Mike) will be deleted from the Manager relation. Subsequently, p_1 and p_4 will execute, first inserting the tuple Manager(Mike) and then deleting this tuple from the Manager relation. Processing of p_1 and p_4 will continue but an operational fixpoint is not reached.

Example 2

The initial EDB = {Employee(Mike).} and the productions are as follows:

(p p_1 (Employee(X)) → (**make** GoodWorker(X)))
(p p_2 (Employee(X), GoodWorker(X)) →
(**make** Manager(X)))
(p p_3 (Employee(X), ¬ GoodWorker(X)) →
(**make** PoorWorker(X)))

If productions execute in the sequence p_1 followed by p_2, then the final EDB will contain the set of tuples {Employee(Mike). GoodWorker(Mike). Manager(Mike).}. If the execution sequence were p_3 followed by p_1 and p_2, then, the final EDB would include the tuples, {Employee(Mike). GoodWorker(Mike). Manager(Mike). PoorWorker(Mike).}. Thus, in this case there are two fixpoints.

A Corresponding Logical Theory

In order to understand the shortcomings of the operational semantics, and to define a *correct semantics*, we define how a logical theory can be obtained from the productions in the PS program.

Definition

Every production that has a **make** action in its **consequent** corresponds to a rule of an intensional database (IDB).

For example, the following production:
(p p_1 (P(x), ¬ Q(x)) → (**make** R(x))) corresponds to the rule
P(x), ¬ Q(x) → R(x).

Every production that has a **remove** action in its consequent corresponds to an integrity constraint (IC). General forms of integrity constraints and their treatment are discussed in Kowalski and Sadri 1989; we delay a detailed discussion of constraints to a later section 4.

Definition

Every production that has **(remove P(x))** in the consequent is a constraint. The literal P(x) is retracted literal to restore consistency.

Thus, the following production:
(p p_2 (P(x), ¬ Q(x)) → (**remove** P(x)))
corresponds to the following IC:
P(x), ¬ Q(x) → .
P(x) is retracted to maintain consistency and the database must not prove
∀x P(x) ∧¬Q(x).

The meaning of treating some productions as integrity constraints will be discussed in section 4.

The following logical theory will be obtained, corresponding to the productions of Example 1:
EDB = { Employee(Mike). GoodWorker(Mike). }
IDB = {Employee(X), GoodWorker(X) →
Manager(X).
Manager(X) → HasOffice(X) .
Employee(X), HasOffice(X) →PoorWorker(X).}
IC = {Manager(X), PoorWorker(X) → . }

We can see that the logical theory of the PS is inconsistent with the constraints. Thus, there can be no model for the theory which is also a model for the constraints (Lloyd 1987). In the case of Example 2, there is a negative literal in the antecedent of a rule. With Horn theories (including stratified theories), the modus ponens rule of inference is insufficient to prove negative information. To do so one uses the closed world rule of inference or the closed world assumption (CWA). The correct interpretation, based on the CWA, is

that a tuple, $<a_1, a_2, \ldots a_n>$ is not true if and only if the IDB ∪ EDB viewed as a logical theory cannot prove $P(a_1, a_2, \ldots a_n)$. In our example, it is clear that the theory can prove GoodWorker(Mike), and ¬ GoodWorker(Mike) cannot be proved. Thus, the interpretation of negative literals in the operational semantics of OPS5 is incorrect.

3. Stratified Production Systems

We identify a class of stratified PS programs, and we define an operational semantics for these programs. Stratification is an extension of Horn programs to more general Horn programs that allow negative literals in the antecedent of a rule; recall that productions in OPS5 have the same feature. Our research draws upon existing research in stratified databases (Apt, Blair and Walker 1988).

The Operational Semantics

A stratified production system program is viewed as a stratified program, and comprises an extensional database (EDB) of facts, a stratified intensional database (IDB) of rules, and a stratified set of integrity constraints (IC). Both rules and integrity constraints correspond to productions in PS.

The initial PS program comprises the following:

(1) an intensional database of rules (IDB), where each rule corresponds to a production that has a **make** action as its consequent,

(2) a set of integrity constraints (IC), where each integrity constraint corresponds to a production that has a **remove** action as its consequent, and

(3) an initial extensional database of facts (EDB_0).

In the operational semantics, there must exist a partition so that PS is a stratified database. Thus,

$$PS = PS_0 \cup PS_2 \ldots \cup PS_n.$$

Each of the partitions PS_i comprises a set of rules IDB_i, and a set of integrity constraints IC_i, each of which may be possibly empty. Partition PS_0 comprises the initial extensional database EDB_0; IDB_0 and IC_0 are null. The following conditions hold for the stratification:

(1) $IDB = IDB_1 \cup \ldots \cup IDB_n$

(2) $IC = IC_1 \cup \ldots \cup IC_n$

(3) For every positive literal in the body of a rule in IDB_i, or in the body of an integrity constraint in IC_i, the definition of that literal must be contained within $\bigcup_{j \leq i} IDB_j$. The definition of a literal is all productions in which the literal occurs in the **make** action.

(4) For every positive literal in the body of a rule in IDB_i, or in the body of an integrity constraint in IC_i, all *other* integrity constraints where the literal occurs in the **remove** action must be contained within $\bigcup_{j < i} IC_j$.

(5) For every negative literal in a rule in IDB_i, or in an integrity constraint in IC_i, the definition of that literal must be contained within $\bigcup_{j < i} IDB_j$.

(6) For every negative literal in a rule in IDB_i, or in an integrity constraint in IC_i, all integrity constraints where the literal occurs in the **remove** action must be contained within $\bigcup_{j \leq i} IC_j$.

The Operator T_{PS}

Once such a partition has been obtained for the stratified PS program, then, for each partition PS_i, we define an operator T_{PS_i} as follows:

Definition

U_{PS} is a Herbrand universe for the stratified PS comprising predicates (**make** P), (**remove** P), where P ranges over all propositional variables in the PS.

If we consider a PS program where the propositional variables are the set {A, B, C}, then U_{PS} is the set of predicates {(**make A**), (**remove A**), (**make B**), (**remove B**),(**make C**), (**remove C**)}.

In the case of predicate variables, we will use relations to represent U_{PS}, two for each predicate symbol. Thus, for predicate P, relation makeP will contain all tuples P(.,.,...) which are added by some action (**make** P(.,.,...)) and relation removeP will contain all tuples P(.,.,...) which are removed by some action (**remove** P(.,.,...)).

Definition

For each proposition P (or ground positive literal P(.,.,...)), in EDB_0, we replace it with (**make** P), (or we add the tuple P(.,.,...) to the

relation makeP).

Definition

The interpretation of the predicates of U_{PS} (for the propositional variable P), is defined as follows:

(1) (**make** P) $\wedge \neg$ (**remove** P) $\models$ P

(2) (**make** P) $\wedge$ (**remove** P) $\models \neg$ P

(3) $\neg$ (**make** P) $\wedge \neg$ (**remove** P) $\models \neg$ P

(4) $\neg$ (**make** P) $\wedge$ (**remove** P) is not allowed in the operational semantics and represents an inconsistency.

Definition

T_{PS_i} is a mapping from a subset of U_{PS} to U_{PS}, such that for any rule in IDB_i or any integrity constraint in IC_i, if there exists an interpretation of EDB_{i-1}, as described above, such that the literals in the antecedent of the corresponding production are true, then, the operator adds the **action** in the head of the production to EDB_i.

Each EDBi is computed as follows:

$$EDB_1 = EDB_0 \cup T_{PS_1} \uparrow \omega\ (EDB_0)$$
$$EDB_2 = EDB_1 \cup T_{PS_2} \uparrow \omega\ (EDB_1)$$

$$EDB_n = EDB_{n-1} \cup T_{PS_n} \uparrow \omega\ (EDB_{n-1})$$

Obtaining an Operational Fixpoint for T_{PS_i}

Processing for each partition PS_i should terminate when an operational fixpoint is reached, i.e., when there are no longer any satisfied productions that can update EDB_i. Processing for PS terminates when the operational fixpoint for PS_n is reached and EDB_n is computed.

Theorem

$T_{PS_i} \uparrow \omega\ (EDB_{i-1})$ is a fixpoint for $PS_i = IDB_i \cup IC_i$.

We refer the reader to (Rasc89) for the proof.

Example 3

Consider a PS whose initial EDB has two tuples, {Employee(Mike). GoodWorker(Mike).}, and the following set of productions:

(p p_1 (Employee(X), GoodWorker(X)) →
(**make** Manager(X)))

(p p_2 (Employee(X)) → (**make** HasOffice(X)))

(p p_3 (PoorWorker(X), HasOffice(X)) →
(**remove** HasOffice(X)))

(p p_4 (Employee(X), HasOffice(X)) →
(**make** Manager(X)))

The stratification conditions will place productions p_3 in a higher stratum, say PS_2, while the other three productions are placed in PS_1. The operational fixpoint EDB_1 is as follows: {makeEmployee(Mike). makePoorWorker(Mike). makeHasOffice(Mike). removeHasOffice(Mike).} EDB_1 is interpreted to prove $\neg$ HasOffice(Mike) and Manager(Mike) will not be an answer.

4. Maintaining Consistency with Integrity Constraints

We now try to understand the meaning of maintaining consistency with respect to integrity constraints in a stratified PS program. Informally, a database must satisfy its integrity constraints as it changes over time. Usually, an update to the database (more precisely an update to facts in the EDB) may cause the violation of an integrity constraint; such updates are rejected or modified. Sometimes, the database itself, i.e., the facts and the rules may be inconsistent with the constraints and must be modified, to maintain consistency. In Kowalski and Sadri 1989, such a method is presented for transforming a theory which is inconsistent with respect to a set of integrity constraints into a transformed theory which is consistent with the constraints. The transformation is syntactic. Although the transformation works for more general constraints, we restrict the discussion to *denial* constraints of the form

A(t), Conj→.

where Conj is a conjunction of positive or negative literals (and may be empty). The denial, or the body of the constraint, should not be true in the database. The atom A is the *retractable* atom that restores consistency, if the constraint is violated.

Thus, if we consider the theory {C. C → A} and the constraint A, ¬B →, where the retractable atom is A, then, the theory is inconsistent with the constraint. However, the transformed theory {C. C, B →A} is consistent with the constraint.

If we apply a similar transformation to the theory obtained from the productions of Example 3, we obtain the following consistent theory:

EDB = {Employee(Mike). GoodWorker(Mike). }

IDB={Employee(X), GoodWorker(X)→Manager(X).
Employee(X) ¬PoorWorker(X)→ HasOffice(X).
Employee(X), HasOffice(X) → Manager(X).}

We see that this theory will not prove Manager(Mike), just as in the operational semantics. In general, we can prove that if the PS

program is stratifiable as defined, then the following hold: (1) the logical theory derived from the stratified PS program, modified to be consistent with the constraints (also derived from the program), is a stratified theory. (2) the operational fixpoint of the stratified PS program is identical to the model for the modified consistent theory that is derived from the program.

5. Summary

A class of stratified PS programs and an operational semantics for these programs have been defined. The theory derived from the productions is modified so that it forms a consistent theory, $\overline{PS}$, with the integrity constraints that are also derived from the productions. The operational fixpoint of PS is identical to the model for $\overline{PS}$. A formal presentation of this research is in Raschid 1989. Future research includes extending the semantics of PS programs to include PS programs that exhibit non–deterministic behaviour when executing productions.

6. References

Apt, K.R., Blair, H.A. and Walker, A. 1988. Towards a Theory of Declarative Knowledge. In (Minker 1988).

Delcambre, L.M.L. and Etheredge, J.N. 1988. A Self–Controlling Interpreter for the Relational Production Language. In Proceedings of the ACM Sigmod International Conference on the Management of Data.

Forgy, C.L. 1981. OPS5 User's Manual, Technical Report CMU–CS–81–135, Carnegie–Mellon University.

Forgy, C.L. 1982. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence* (19).

Hayes–Roth, F. 1985. Rule Based Systems. *Communications of the ACM* (28) 9.

Kowalski, R. and Sadri, F. 1989. Knowledge Representation without Integrity Constraints. Technical Report, Imperial College, London, England.

Lloyd, J.W. 1987. *Foundations of Logic Programming,* Second, Extended Edition. Springer Verlag.

Maindreville, C. de and Simon, E. 1988. A Production Rule Based Approach to Deductive Databases. In Proceedings of the Fourth International Conference on Data Engineering.

Minker. J., ed. 1988. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers, Inc.

Raschid, L. 1989. Defining a Semantics for Production Systems based on Stratified Databases and Integrity Constraints, Technical Report 89–103, University of Maryland.

Raschid, L., Sellis, T. and Lin, C–C. 1988. Exploiting Concurrency in a DBMS Implementation for Production Systems. In Proceedings of the International Symposium on Databases in Parallel and Distributed Systems.

Sadri, F. and Kowalski, R. 1988. A Theorem–Proving Approach to Database Integrity. In (Minker 1988).

Sellis, T., Lin, C–C. and Raschid, L. 1988. Implementing Large Production Systems in a DBMS Environment: Concepts and Algorithms. In Proceedings of the ACM Sigmod International Conference on the Management of Data.

Simon, E. and Maindreville, C. de, 1988. Deciding Whether a Production Rule is Relational Computable. In IProceedings of the International Conference on Database Theory.

Widom, J. and Finkelstein, S.J. 1989. A Syntax and Semantics for Set–Oriented Production Rules in Relational Database Systems, IBM Research Report, IBM Almaden Research Center.