PARTIAL EVALUATION IN LOGIC PROGRAMMING

J. W. LLOYD AND J. C. SHEPHERDSON

This paper gives a theoretical foundation for partial evaluation in logic programming. Let $P$ be a normal program, $G$ a normal goal, $A$ a finite set of atoms, and $P'$ a partial evaluation of $P$ wrt $A$. We study, for both the declarative and procedural semantics, conditions under which $P'$ is sound and complete wrt $P$ for the goal $G$. We identify two relevant conditions, those of closedness and independence. For the procedural semantics, we show that, if $P' \cup \{G\}$ is $A$-closed and $A$ is independent, then $P'$ is sound and complete wrt $P$ for the goal $G$. For the declarative semantics, we show that, if $P' \cup \{G\}$ is $A$-closed, then $P'$ is sound wrt $P$ for the goal $G$. However, we show that, unless strong conditions are imposed, we do not have completeness for the declarative semantics. A practical consequence of our results is that partial evaluators should enforce the closedness and independence conditions.

1. INTRODUCTION

Partial evaluation is an optimization technique dating back to Kleene's s-m-n theorem [15]. It was explicitly introduced into computer science by Futamura [11], although aspects of partial evaluation already existed at that time in compiler optimization. The idea was studied intensively in Sweden (see [1], for example) and Russia (see [8]) in the 1970s.

Komorowski [16] introduced partial evaluation into logic programming in 1981. After several years of neglect, the importance of partial evaluation as an optimiza-
tion technique was realized and there is now substantial and growing interest in it [2, 5, 6, 9, 10, 13, 14, 18, 21, 27]. Applications of partial evaluation, especially to metaprogramming, are described in [7], [12], [19], [22], [26], [28], and [29], for example. A collection of recent papers and an extensive bibliography [23] on partial evaluation is contained in [3].

In logic programming terms, partial evaluation can be described as follows. Given a program \( P \) and a goal \( G \), partial evaluation produces a new program \( P' \), which is \( P \) "specialized" to the goal \( G \). The intention is that \( G \) should have the same (correct and computed) answers wrt \( P \) and \( P' \), and that \( G \) should run more efficiently for \( P' \) than for \( P \). The basic technique for obtaining \( P' \) from \( P \) is to construct "partial" search trees for \( P \) and suitably chosen atoms as goals, and then extract \( P' \) from the definitions associated with the leaves of these trees.

Many of the above papers report the results of partial evaluation in a number of application areas and, generally, show that the use of partial evaluation can result in great gains in efficiency. In particular, when applied to metaprogramming, partial evaluation can be used to "compile away" the layers of software to produce a much more efficient system.

However, while it is clear that many logic programmers are making practical use of partial evaluation, there does not appear to be any study of its foundations. This paper provides such a foundation. As our results show, partial evaluation can be given a firm theoretical basis, which not only provides a justification for the published techniques, but also indicates further research directions.

The main foundational questions are concerned with the soundness and completeness of partial evaluation. Soundness of the partially evaluated program \( P' \) wrt the original program \( P \) and goal \( G \) for the declarative (respectively, procedural) semantics means that correct (computed) answers for \( G \) and \( P' \) are correct (computed) answers for \( G \) and \( P \). Completeness is the converse of this. We show that, for definite programs and goals, partial evaluation is always sound, but may not be complete. We isolate a closedness condition which ensures completeness. The closedness condition involves a simple syntactic check on the partially evaluated program and goal. For normal programs and goals (that is, negation is allowed), we show that partial evaluation is not only, in general, incomplete, but is also, in general, unsound. We show how the closedness condition, together with an independence condition, ensures both soundness and completeness (at least for the procedural semantics—the results for the declarative semantics are less satisfactory). It is thus apparent that the closedness and independence conditions are important ones, and a practical consequence of our results is that partial evaluators need to enforce these conditions.

An outline of this paper is as follows. Section 2 gives the basic concepts of partial evaluation and illustrates the important ideas with some examples. Because of the incompleteness, in general, of SLDNF-resolution, we are forced to consider the theory of partial evaluation for the declarative and procedural semantics separately. Section 3 contains the results for the declarative semantics, and Section 4 contains the results for the procedural semantics. Section 5 concludes the paper with some remarks about future research directions.

We assume the reader is familiar with the standard theoretical results of logic programming, which are contained in [20]. The notation and terminology are consistent with [20].
2. BASIC CONCEPTS OF PARTIAL EVALUATION

This section contains the definitions of the basic concepts of partial evaluation and some illustrative examples. We begin with the definitions of definite (and of normal) programs and goals, as presented in [20]. Throughout, we assume that there is some fixed underlying first order language and that the symbols in programs, goals, and other first order formulas appear in this language.

Definition. A program clause (definite program clause) is a clause of the form
\[ A \leftarrow L_1, \ldots, L_n \]
where \( A \) is an atom and \( L_1, \ldots, L_n \) are literals (atoms).

Definition. A normal program (definite program) is a finite set of program clauses (definite program clauses).

Definition. A normal goal (definite goal) is a clause of the form
\[ \leftarrow L_1, \ldots, L_n \]
where \( L_1, \ldots, L_n \) are literals (atoms).

Definition. The definition of a predicate symbol \( p \) in a normal program \( P \) is the set of all program clauses in \( P \) which have \( p \) in their head.

The following definitions and notations are also contained in [20].

Definition. Let \( J \) be a preinterpretation of a first order language, \( V \) a variable assignment wrt \( J \), and \( A \) an atom. Suppose \( A \) is \( p(t_1, \ldots, t_n) \), and \( d_1, \ldots, d_n \) in the domain of \( J \) are the term assignments of \( t_1, \ldots, t_n \) wrt \( J \) and \( V \). We call \( A_{J,V} = p(d_1, \ldots, d_n) \) the \( J \)-instance of \( A \) wrt \( V \). We put \( [A]_J = \{ A_{J,V} : V \text{ is a variable assignment wrt } J \} \).

If \( P \) is a normal program, then the completion of \( P \) is denoted by \( \text{comp}(P) \). The mapping \( T^*_p \) from the lattice of interpretations based on some preinterpretation \( J \) to itself is defined as follows.

Definition. Let \( J \) be a preinterpretation of a normal program \( P \), and \( I \) an interpretation based on \( J \). Then \( T^*_p(I) = \{ A_{J,V} : A \leftarrow L_1, \ldots, L_n \in P, V \text{ is a variable assignment wrt } J, \text{ and } L_1 \lor \cdots \lor L_n \text{ is true wrt } I \text{ and } V \} \).

We will find it convenient to denote an atom \( p(t_1, \ldots, t_n) \) by \( p(\bar{t}) \) and a \( J \)-instance \( p(d_1, \ldots, d_n) \) by \( p(\bar{d}) \). Throughout, \( Q \), subscripted or not, denotes a conjunction of literals. If \( \theta \) and \( \phi \) are answers to a goal \( G \), we say \( \theta \) includes \( \phi \) if there exists a substitution \( \gamma \) such that \( \phi \) and \( \theta \gamma \) have the same effect on all variables in \( G \).

For the definitions of SLDNF-derivation, SLDNF-refutation and SLDNF-tree, we refer the reader to [20]. In fact, it will be convenient in the following to use slightly more general definitions of SLDNF-derivation and SLDNF-tree than are given there. In [20], an SLDNF-derivation in either infinite, successful, or failed.
Here we will also allow it to be *incomplete*, in the sense that any point we are allowed to simply not select any literal and terminate the derivation. Similar remarks apply to the definition of SLDNF-tree employed here.

Next we give the definition of an SLD-derivation (and of an SLD-tree) wrt a *normal* program and goal.

**Definition.** Let $P$ be a normal program and $G$ a normal goal. An *SLD-derivation* (SLD-tree) of $P \cup \{G\}$ is an SLDNF-derivation (SLDNF-tree) of $P \cup \{G\}$ such that only positive literals are selected.

A concept that will be needed in the definition of partial evaluation is that of a resultant, which we now define.

**Definition.** A resultant is a first order formula of the form $Q_1 \leftarrow Q_2$, where $Q_i$ is either absent or a conjunction of literals ($i = 1, 2$). Any variables in $Q_1$ or $Q_2$ are assumed to be universally quantified at the front of the resultant.

**Definition.** Let $P$ be a normal program, $G$ a normal goal $\leftarrow Q$, and $G_\theta = G, G_1, \ldots, G_n$ an SLDNF-derivation of $P \cup \{G\}$, where the sequence of substitutions is $\theta_1, \ldots, \theta_n$ and $G_n$ is $\leftarrow Q_n$. Let $\theta$ be the restriction of $\theta_1, \ldots, \theta_n$ to the variables in $G$. Then we say the derivation has length $n$ with *computed answer* $\theta$ and *resultant* $Q\theta \leftarrow Q_n$. (If $n = 0$, the resultant is $Q \leftarrow Q$.)

A resultant is not in general a clause, because the $Q$ on the left stands for a *conjunction* of literals. For an SLD-derivation, the usual soundness theorem shows that (the universal closure of) a resultant is a logical consequence of the clauses used. This is familiar in the special case when $G_n$ is empty and we have proved the universal closure of $Q\theta$. The resultant encapsulates most of the information about the result of the derivation. It does not include all, because it does not give the original goal $G$ and the computed answer $\theta$, but only the effect $G\theta$ of applying $\theta$ to $G$. But, if you are given $G$ as well as the resultant, then you can find $\theta$. We shall constantly use the evident fact that the resultant of a derivation carries all the information needed to find the resultant of the next step, i.e., knowledge of the original goal is unnecessary. A resultant is particularly significant when the original goal $G$ consists of a single atom $\leftarrow A$, because the resultant is then the program clause

$A\theta \leftarrow Q_n$

and, by Lemma 4.5 below, the original derivation from the goal $\leftarrow A$ is equivalent to (i.e., has the same resultant as) a single SLD-resolution step with this new clause. Lemma 4.12 states that the same is true for goals of the form $\leftarrow A\phi$.

Next we give the definition of a partial evaluation of a normal program wrt a set of atoms.

**Definition.** Let $P$ be a normal program, $A$ an atom, and $T$ an SLDNF-tree for $P \cup \{\leftarrow A\}$. Let $G_1, \ldots, G_r$ be (nonroot) goals in $T$ chosen so that each nonfailing branch of $T$ contains exactly one of them. Let $R_i$ ($i = 1, \ldots, r$) be the resultant of the derivation from $\leftarrow A$ down to $G_i$ given by the branch leading to $G_i$. Then the set of clauses $R_1, \ldots, R_r$ is called a *partial evaluation* of $A$ in $P$. 
If $A = \{A_1, \ldots, A_s\}$ is a finite set of atoms, then a partial evaluation of $A$ in $P$ is the union of partial evaluations of $A_1, \ldots, A_s$ in $P$.

A partial evaluation of $P$ wrt $A$ is a normal program obtained from $P$ by replacing the set of clauses in $P$ whose head contains one of the predicate symbols appearing in $A$ (called the partially evaluated predicates) with a partial evaluation of $A$ in $P$.

A partial evaluation of $P$ wrt $A$ using SLD-trees is a partial evaluation of $P$ wrt $A$ in which all the SLDNF-trees used for the partial evaluation are actually SLD-trees (i.e., no negation as failure steps are allowed during the partial evaluation process).

We now illustrate these definitions with an example.

Example. Consider the definite program $P$ given by

- $\text{grandmother}(x, y) \leftarrow \text{mother}(x, z), \text{parent}(z, y)$
- $\text{grandfather}(x, y) \leftarrow \text{father}(x, z), \text{parent}(z, y)$
- $\text{parent}(x, y) \leftarrow \text{mother}(x, z), \text{parent}(z, y)$
- $\text{parent}(x, y) \leftarrow \text{father}(x, y)$
- $\text{mother}(\text{Sue}, \text{Simon}) \leftarrow$
- $\text{mother}(\text{Sue}, \text{Patrick}) \leftarrow$
- $\text{mother}(\text{Monica}, \text{Sue}) \leftarrow$

Together with a definition for $\text{father}$. Let $G$ be the goal $\leftarrow \text{grandmother}(x, \text{Simon})$. Then a partial evaluation of $P$ wrt $\{\text{grandmother}(x, \text{Simon})\}$ produces the following definition for $\text{grandmother}$, where the atom selected by the computation rule was always the leftmost atom, unless the predicate in the atom was $\text{father}$, in which case the derivation was terminated:

- $\text{grandmother}(\text{Sue}, \text{Simon}) \leftarrow \text{father}(\text{Simon}, \text{Simon})$
- $\text{grandmother}(\text{Sue}, \text{Simon}) \leftarrow \text{father}(\text{Patrick}, \text{Simon})$
- $\text{grandmother}(\text{Monica}, \text{Simon}) \leftarrow$
- $\text{grandmother}(\text{Monica}, \text{Simon}) \leftarrow \text{father}(\text{Sue}, \text{Simon})$

The program $P'$, consisting of this definition for $\text{grandmother}$ together with the one for $\text{father}$, can then be used to answer goals of the form $\leftarrow \text{grandmother}(x, \text{Simon})$. The soundness of $P'$ is obvious by the soundness of resolution. The completeness of $P'$ for goals of the form $\leftarrow \text{grandmother}(x, \text{Simon})$ is not so obvious, but follows from our results below.

To motivate the next definition, we consider an example where the partial evaluation of a program is a not complete wrt goals of a certain form.

Example. Let $P$ be the definite program

- $p \leftarrow q(a)$
- $q(x) \leftarrow$
and suppose that we partially evaluate $P$ wrt $\{q(b)\}$ to produce the program $P'$:

$$
p \leftarrow q(a)
$$

$$
q(b) \leftarrow
$$

Now $P \cup \{\leftarrow p\}$ has an SLD-refutation, but $P' \cup \{\leftarrow p\}$ does not.

To ensure completeness, it suffices to put a closedness condition on the program resulting from the partial evaluation. This condition, given below, would require for the above example that every atom containing $q$ which occurs in the body of a clause in $P'$ or a goal should be an instance of $q(b)$. Intuitively, if we “specialize” the definition of $q$ to $q(b)$, then we cannot expect to be able to correctly answer calls to $q$ which are not instances of $q(b)$.

With this motivation, we present the closedness condition.

**Definition.** Let $S$ be a set of first order formulas and $A$ a finite set of atoms. We say $S$ is $A$-closed if each atom in $S$ containing a predicate symbol occurring in an atom in $A$ is an instance of an atom in $A$.

In the presence of negation, the following example shows that, without the closedness condition, we do not even have soundness.

**Example.** Let $P$ be the normal program

$$
p \leftarrow \neg q
$$

$$
q \leftarrow r(a)
$$

$$
r(x) \leftarrow
$$

If we partially evaluate $P$ wrt $\{r(b)\}$, we obtain the normal program $P'$:

$$
p \leftarrow \neg q
$$

$$
q \leftarrow r(a)
$$

$$
r(b) \leftarrow
$$

Now $P' \cup \{\leftarrow p\}$ has an SLDNF-refutation, but $P \cup \{\leftarrow p\}$ does not.

We define here a condition on $A$ which will be needed for some later theorems.

**Definition.** Let $A$ be a finite set of atoms. We say $A$ is independent if no pair of atoms in $A$ have a common instance.

As the results of Sections 3 and 4 show, the closedness condition (together with independence, in some cases) is sufficient to obtain the soundness of partial evaluation for both the declarative and the procedural semantics.

### 3. RESULTS FOR THE DECLARATIVE SEMANTICS

This section contains the basic results for partial evaluation for the declarative semantics. We take the declarative semantics as given by the completion of a normal program and correct answers wrt the completion. (See [20] for the details.)
The first soundness theorem shows that, provided the closedness condition is satisfied, an answer which is correct wrt the partial evaluation of a program is also correct wrt the original program.

**Theorem 3.1.** Let $P$ be a normal program, $W$ a closed first order formula, $A$ a finite set of atoms, and $P'$ a partial evaluation of $P$ wrt $A$ such that $P' \cup \{W\}$ is $A$-closed. If $W$ is a logical consequence of $\text{comp}(P')$, then $W$ is a logical consequence of $\text{comp}(P)$.

**Proof.** We give the proof for the case when $A$ consists of a single atom $p(i)$. The extension to the general case is straightforward.

Let $J$ be a preinterpretation for the language underlying $P$, and $I$ an interpretation based on $J$ such that $I$, together with the identity relation assigned to $=$, is a model for $\text{comp}(P)$. Let $I'$ be $\{q(\vec{d}) \in I : q \neq p\} \cup \{p(\vec{d}) \in I : p(\vec{d}) \in \{p(i)\}_J\}$.

First we show that $I'$ is a fixpoint for $T^J_P$.

(i) $T^J_P(I') \subseteq I'$: First note that $T^J_P(I') = T^J_P(I)$, since $P'$ is $\{p(i)\}$-closed. Now $P'$ is a logical consequence of $\text{comp}(P)$. Hence $I$ is also a model for $P'$. That is, $T^J_P(I) \subseteq I$ and so $T^J_P(I') \subseteq I$. Also, $p(\vec{d}) \in T^J_P(I')$ implies $p(\vec{d}) \in \{p(i)\}_J$. Hence $T^J_P(I') \subseteq I'$.

(ii) $I' \subseteq T^J_P(I')$: We first prove that $I' \subseteq T^J_P(I)$ by induction on the number, $n$, of goals which are predecessors of goals corresponding to resultants in the SLDNF-tree used to construct $P'$.

$n = 0$: In this case, $P \cup \{\neg p(\vec{t})\}$ has a finitely failed SLDNF-tree. Hence, by the soundness of the negation as failure rule (Theorem 15.4 of [20]), $\forall \vec{v} \left(\neg p(\vec{t})\right)$ is a logical consequence of $\text{comp}(P)$. Thus $I' = \{q(\vec{d}) \in I : q \neq p\}$. Since $P'$ is the same as $P$ except that the definition for $p$ is deleted, it follows that $I' \subseteq T^J_P(I)$.

$n = 1$: Since $P$ and $P'$ differ only on the definition for $p$, we can confine attention to $J$-instances in $I'$ of the form $p(\vec{d})$. Let $p(\vec{d}) \in I'$. Hence $p(\vec{d}) = p(\vec{t})_I$, $V$ for some variable assignment $V$. Also $p(\vec{d}) \in I$ and $I = T^J_P(I)$. Hence there is a clause $p(\vec{d}) \leftarrow Q$ in $P$ such that $p(\vec{d}) = p(\vec{t})_I$ and $Q$ is true wrt $I$ and $V$. By Lemma 15.2(a) of [20], $p(\vec{t})$ and $p(\vec{s})$ are unifiable with mgu $\theta$, say. By Lemma 15.2(b) of [20], $p(\vec{d}) = p(\vec{s})_I V$ and $Q \theta$ is true wrt $I$ and $V$. Note that the children of the goal $\leftarrow p(\vec{t})$ in the SLDNF-tree used to construct $P'$ must either correspond to a resultant or else be the root of a finitely failed SLDNF-tree. Hence, by the soundness of the negation as failure rule, $p(\vec{t}) \theta \leftarrow Q \theta$ is in $P'$. Thus $p(\vec{d}) \in T^J_P(I)$.

Induction step: In the SLDNF-tree used to construct $P'$, choose a goal $G$ which is a predecessor of a goal corresponding to a resultant and which has maximum depth amongst such goals. Let $P''$ be the normal program obtained by partial evaluation of $P$ wrt $\{p(\vec{t})\}$ using the resultant corresponding to $G$, together with all the previous resultants not corresponding to successors of $G$.

By the induction hypothesis, we have that $I' \subseteq T^J_P(I)$. Thus it suffices to show that $T^J_P(I) \subseteq T^J_P(I)$. Let $p(\vec{d}) \in T^J_P(I)$. Hence $p(\vec{d}) = p(\vec{t})_I$, $V$ for some clause $p(\vec{r}) \leftarrow Q$ in $P''$ and variable assignment $V$. Now, if $p(\vec{r}) \leftarrow Q$ is not the resultant in $P''$ corresponding to $G$, we are finished. Otherwise, suppose $Q$ is $Q_1 \land L \land Q_2$, where $L$ is the selected literal.

Suppose first that $L$ is a positive literal $A$. Now $A_{I,V} \in I$ and $I \subseteq T^J_P(I)$. Hence there exists a clause $q(\vec{u}) \leftarrow F$ in $P$ such that $A_{I,V} = q(\vec{u})_I$ and $F$ is true wrt $I$.
and $V$. By Lemma 15.2(a) of [20], $A$ and $q(\bar{u})$ are unifiable with mgu $\phi$, say. By Lemma 15.2(b) of [20], $p(\bar{d}) = (p(\bar{r})\phi)_{I, V}$ and $(Q_1 \land F \land Q_2)\phi$ is true wrt $I$ and $V$. By the soundness of the negation as failure rule, $p(\bar{r})\phi \leftarrow (Q_1 \land F \land Q_2)\phi$ is in $P'$. Thus $p(\bar{d}) \in T^I_{P'}(I)$.

If $L$ is a negative literal, then $p(\bar{r}) \leftarrow Q_1 \land Q_2$ is in $P'$ and hence $p(\bar{d}) \in T^I_{P'}(I)$.

This completes the induction argument.

Now, since $P'$ is $(p(\bar{r}))$-closed, we have that $I' \subseteq T^I_{P'}(I')$, and this completes the proof that $I'$ is a fixpoint of $T^i_{P'}$.

Since $W$ is a logical consequence of $\text{comp}(P')$ and $I'$ is a fixpoint for $T^i_{P'}$, we have that $W$ is true wrt $I'$. But $W$ is $(p(\bar{r}))$-closed and hence $W$ is true wrt $I$. Thus $W$ is a logical consequence of $\text{comp}(P)$.

Note that we need to consider the cases $n = 0$ and $n = 1$ separately in the proof because, in the definition of partial evaluation, resultants can only come from nonroot goals. Theorem 3.1 has some direct consequences, which are given by the next theorem.

**Theorem 3.2.** Let $P$ be a normal program, $G$ a normal goal, $A$ a finite set of atoms, and $P'$ a partial evaluation of $P$ wrt $A$ such that $P' \cup \{G\}$ is $A$-closed. Then the following hold:

(a) If $\theta$ is a correct answer for $\text{comp}(P') \cup \{G\}$, then $\theta$ is a correct answer for $\text{comp}(P) \cup \{G\}$.

(b) If $\theta$ is a computed answer for $P' \cup \{G\}$, then $\theta$ is a correct answer for $\text{comp}(P) \cup \{G\}$.

(c) If $P' \cup \{G\}$ has a finitely failed SLDNF-tree, then $G$ is a logical consequence of $\text{comp}(P)$.

**PROOF.** Part (a) follows directly from Theorem 3.1. Part (b) follows from Theorem 15.6 of [20] and (a). Part (c) follows from Theorem 15.4 of [20] and Theorem 3.1. □

Theorem 3.2(b) is an important one. It shows that partial evaluation is sound, in the sense that, for the goal $G$, every answer computed from the partially evaluated program $P'$ is correct wrt the declarative semantics for $P$ given by its completion.

Next we turn to the converse of Theorems 3.1 and 3.2. The following example shows that, under the conditions of Theorem 3.1, if $W$ is a logical consequence of $\text{comp}(P)$ we do not necessarily have that $W$ is a logical consequence of $\text{comp}(P')$.

**Example.** Let $P$ be the stratified normal program

\begin{align*}
p & \leftarrow \neg q \\
q & \leftarrow r, \neg s \\
r & \leftarrow s \\
s & \leftarrow r
\end{align*}
If we partially evaluate $P$ wrt \{r, s\}, we can obtain the program $P'$:

$$
p \leftarrow \neg q
q \leftarrow r, \neg s
r \leftarrow r
s \leftarrow s
$$

Now $p$ is a logical consequence of $\text{comp}(P)$, but $p$ is not a logical consequence of $\text{comp}(P')$.

As we shall see in the next section, there is a satisfactory completeness result for the procedural semantics. Hence the essence of the problem here is the incompleteness, in general, of SLDNF-resolution. Under assumptions which ensure the completeness of SLDNF-resolution (for example, allowedness and strictness [4, 17]), we can use the completeness result of the next section (Theorem 4.3) to obtain a completeness result for partial evaluation for the declarative semantics [that is, a converse of Theorem 3.2(a)]. Without these conditions, there seems little hope of such a result.

4. RESULTS FOR THE PROCEDURAL SEMANTICS

In this section we prove that, if $P'$ is a partial evaluation of $P$, then, under the closedness and independence conditions, the programs $P$ and $P'$ are computationally equivalent, i.e., a goal succeeds (finitely fails) using $P'$ iff it succeeds (finitely fails) using $P$. We treat the simpler case of SLD-resolution first (Theorems 4.1 and 4.2) because the argument is simpler, the results are better than in the case of SLDNF-resolution (Theorem 4.3), and their proofs are used in the proof of the latter.

4.1. Basic Lemmas

Although the arguments follow the obvious line, the full proofs are longer than expected. This is particularly true of the crucial Lemmas 4.11 and 4.12 below, which simply justify the use of a derived clause. One reason for the length of the proofs is that we need to generalize to arbitrary derivations, results which are well known for the case of derivations ending in the empty goal. One or two of them already appear in the literature in the form we need, but for the sake of completeness we give proofs of all the lemmas needed. The reader is advised to proceed now to Section 4.2 and read the proofs of Theorems 4.1, 4.2 and 4.3, coming back to the present section when it becomes clear why these lemmas are needed.

We will make use of the lifting lemma of resolution to justify a more general form of resolution, where different substitutions may be applied to the goal and the clause. What we want to do is the following:

From the goal $\leftarrow Q_1, A, Q_2$ and the clause $A' \leftarrow Q'$, for which there exist substitutions $\theta$ and $\theta'$ such that $A\theta = A'\theta'$, derive the goal $\leftarrow Q_1\theta, Q'\theta', Q_2\theta$. 
We call $\theta$ a goal substitution and $\theta'$ a clause substitution. This simplifies our proofs because it avoids the need to take variants of clauses and to prove that substitutions are mgu; the rather tricky arguments of the next lemma are done once and for all, instead of being repeated with each application.

**Definition.** Let $P$ be a normal program and $G$ a normal goal. A GSLD-derivation (G for generalized) of $P \cup \{G\}$ is like an SLD-derivation of $P \cup \{G\}$, except that the basic resolution step (called a GSLD-resolution step) is the more general one above.

**Definition.** Let $P$ be a normal program, $G$ a normal goal $\leftarrow Q$, and $G_0 = G, G_1, \ldots, G_n$ a GSLD-derivation of $P \cup \{G\}$, where the sequence of goal substitutions is $\theta_1, \ldots, \theta_n$ and $G_n$ is $\leftarrow Q_n$. Let $\theta$ be the restriction of $\theta_1, \ldots, \theta_n$ to the variables in $G$. Then we say the GSLD-derivation has length $n$ with computed answer $\theta$ and resultant $Q \theta \leftarrow Q_n$.

**Lemma 4.1 (Lifting).** Let $R$ be the resultant of a GSLD-derivation $G_0, G_1, \ldots, G_n$, using clauses $C_1, \ldots, C_n$. Then $R$ is an instance of the resultant of an SLD-derivation $G_0, G'_1, \ldots, G'_n$, using any variants $C'_1, \ldots, C'_n$ of $C_1, \ldots, C_n$, which are standardized apart, and any mgu.

**Proof.** “Standardized apart” can be defined in a slightly weaker way than in [20]. What is required is that if $\theta_1, \ldots, \theta_n$ is the succession of mgu used, then none of the variables in $C_{i+1}$ occur in $G'_i$ or $G_0 \theta_1, \ldots, \theta_i$, i.e., that they do not occur in the resultant at the beginning of the $(i + 1)$th step (because, writing $G_i = \leftarrow Q_i$ and $G'_i = \leftarrow Q'_i$, this resultant $R'_i$ is $Q_0 \theta_1, \ldots, \theta_i \leftarrow Q'_i$).

The proof is by induction on $n$. If $n = 0$, there is nothing to prove. Now suppose that we have a GSLD-derivation $G_0, \ldots, G_{n+1}$, with clauses $C_1, \ldots, C_{n+1}$, goal substitutions $\theta_1, \ldots, \theta_{n+1}$, clause substitutions $\alpha_1, \ldots, \alpha_{n+1}$, and resultant $R_{n+1}$. Let $R_n$ be the resultant from the first $n$ steps of this derivation. By the induction hypothesis, there is an SLD-derivation $G_0, G'_1, \ldots, G'_n$ with resultant $R_n$, using any suitable variants $C'_1, \ldots, C'_n$ of $C_1, \ldots, C_n$ and mgu $\theta'_1, \ldots, \theta'_n$, and there is a substitution $\delta_n$ such that $R_n = R_n \delta_n$. Hence

$$Q_0 \theta_1, \ldots, \theta_n = Q_0 \theta'_1, \ldots, \theta'_n \delta_n,$$

and

$$G_n = G'_n \delta_n.$$

Suppose that $G'_n$ is $\leftarrow Q_1, A, Q_2$, that $C_{n+1}$ is $A' \leftarrow Q'$, and $A \delta_n$ is the selected atom in $G_n$. Let $\gamma$ be any renaming substitution for $C_{n+1}$ such that $C_{n+1} - C_{n+1} \gamma$ has no variables in common with $R'_{n+1}$. Define the substitution $\phi$ by

$$\phi = \begin{cases} x \delta_n \theta_{n+1} & \text{if } x \text{ occurs in } R'_{n+1}, \\ x' \alpha_{n+1} & \text{if } x \text{ occurs in } C'_{n+1} \text{ and } x = x' \gamma, \text{ where } x' \text{ occurs in } C_{n+1}. \end{cases}$$

Then $\phi$ is a unifier of $A$ and $A \gamma$. If $\theta'_{n+1}$ is any mgu of $A$ and $A \gamma$, then the resultant $R'_{n+1}$ of SLD-resolution using $\theta'_{n+1}$ on $G'_n$ and $C'_{n+1}$ is

$$Q_0 \theta'_1, \ldots, \theta'_{n+1} \leftarrow (Q'_1, Q' \gamma, Q_2) \theta'_{n+1}. $$
Now there exists a substitution $\delta_{n+1}$ such that $\phi = \theta'_{n+1}\delta_{n+1}$. Furthermore,

$$Q_0\theta_1'\ldots\theta'_{n+1}\delta_{n+1} = Q_0\theta_1'\ldots\theta'_{n}\phi$$
$$= Q_0\theta_1'\ldots\theta'_{n}\delta_{n}\theta_{n+1}$$
$$= Q_0\theta_1'\ldots\theta_{n+1}$$

and

$$\leftarrow (Q_1, Q'_1, Q_2)\delta_{n+1} = \leftarrow (Q_1, Q'_1, Q_2)\phi$$
$$= \leftarrow Q_1\delta_n\theta_{n+1}, Q'_1\alpha_{n+1}, Q_2\delta_{n}\theta_{n+1}$$
$$= G_{n+1}.$$ 

Hence $R'_{n+1}\delta_{n+1} = R_{n+1}$. $\square$

From now on, we shall assume all SLDNF-derivations use mgu and clauses standarized apart, in the sense described in the lifting lemma. SLDNF-derivations are usually described without specifying the actual variants of program clauses and the mgu which are to be used. This is justified by the following lemma.

**Lemma 4.2 (Uniqueness)**

(a) If two SLDNF-derivations differ only in the variants of clauses and the mgu which are used, then the resultants are variants of each other.

(b) If an SLDNF-derivation contains at least one resolution step, then every variant of the resultant can be obtained by a suitable choice of the last mgu.

**Proof.** (a): For SLD-derivations, this follows immediately from the lifting lemma. The conclusion for SLDNF-derivations follows because the SLD-resolution steps give unique resultants modulo variants. Also, a negation as failure step is possible on a goal iff it is possible on a variant, and the resultant is unique.

(b): [This is not strictly necessary for our development, but is a conceptual simplification. Instead of saying that the resultants of two derivations (with variants of clauses and mgu unspecified) are the same modulo variants, we can say that they are identical, meaning that the two classes of resultants obtainable coincide.] Suppose that the resultant before the last resolution step is

$$Q_0\theta \leftarrow Q_1, A, Q_2$$

and this last step uses clause $A' \leftarrow Q'$ with mgu $\theta'$ to get the resultant

$$Q_0\theta\theta' \leftarrow (Q_1, Q', Q_2)\theta'.$$

By Lemma 4.3 below, any variant of this can be obtained by applying a substitution $\alpha$ which has an inverse $\alpha^{-1}$. But then $\theta' = (\theta'\alpha)\alpha^{-1}$. Thus $\theta'\alpha$ is also an mgu, and using this instead of $\theta'$ achieves the desired resultant. $\square$

However, some variants cannot be obtained using idempotent mgu. For example, the resultant $p(f(x)) \leftarrow$ cannot be obtained from the goal $\leftarrow p(x)$ and the clause $p(f(y)) \leftarrow$ by any variant of this clause and any idempotent mgu.
Lemma 4.3. Any renaming of an expression can be done by an invertible substitution (i.e., a permutation).

PROOF. Let $\alpha = \{x_1/y_1, \ldots, x_n/y_n\}$ be a renaming substitution for the expression $E$. Form $\alpha'$ by adding to $\alpha$ a binding $y_i/x_i$ for each $y_i$ not in $\{x_1, \ldots, x_n\}$, where the mapping $y_i \rightarrow x_i$ is a bijection from the $y_i$ not in $\{x_1, \ldots, x_n\}$ to the $x_i$ not in $\{y_1, \ldots, y_n\}$. Then $E\alpha = E\alpha'$ because any $y_i$ not in $\{x_1, \ldots, x_n\}$ are not in $E$ either (this is part of the definition of $\alpha$ being renaming), and

$$\bar{\alpha} = \{y_1/x_1, \ldots, y_n/x_n\} \cup \{x_i/y_i : x_i \text{ not in } \{y_1, \ldots, y_n\}\}$$

is a substitution, since $y_1, \ldots, y_n$ are distinct, $x_i$ does not belong to $\{y_1, \ldots, y_n\}$, and $x_i = x_j$ implies that $y_i = y_j$. Clearly, $\bar{\alpha}$ is an inverse of $\alpha'$. 0

The following lemma makes precise a notion of a derivation occurring as a subderivation of another and shows that if two derivations from the same goal have the same resultant, then they are equivalent when used as a subderivation.

Lemma 4.4 (Subderivation)

(a) Let $D = G_0, \ldots, G_n$ and $D' = G_0', \ldots, G'_m$ be SLDNF-derivations with $G_0 \subseteq G'_m$. Then we may define an SLDNF-derivation $D'D = G'_0, \ldots, G'_{m+1}, G'_{m+2}, \ldots, G'_{m+n}$, which starts with $D'$ and continues by selecting in each $G'_{m+i}$ a literal corresponding to the one selected by $D$ in $G_i$ and using a variant of the clause used by $D$ at this step or a corresponding negation as failure step. The resultant of $D'D$ is determined by the resultant of $D'$, by $G_0$, and by the resultant of $D$.

(b) Let $D_1$ (respectively $D_2$) be a derivation from $G_0$, and $D'D_1$ ($D'D_2$) the derivation as defined in part (a). If the resultant of $D_2$ is an instance of the resultant of $D_1$, then the resultant of $D'D_2$ is an instance of the resultant of $D'D_1$.

PROOF. (a): By saying that $G_0 \subseteq G'_m$, we mean that the literals of $G_0$ are a subsequence of those of $G'_m$. Since the place they occupy plays no role in the proof, we simplify the notation by taking $G_0 = \leftarrow Q_0$ and $G'_m = \leftarrow Q_0, Q'$.

Let the resultant $R'$ of $D'$ be $Q_0^0 \theta' \leftarrow Q_0, Q'$, and let $V$ be the variables in $R'$ which are not in $Q_0$. By choosing variants of the clauses and mgu used in $D$, we can assume without loss of generality that none of the variables in $V$ appear in these clauses or mgu. Let the substitutions used in $D$ be $\theta_1, \ldots, \theta_n$, and its resultant $R$ be $Q_0 \theta \leftarrow Q_n$. Then it is clear that the derivation $D'D$ exists. Furthermore, since $\theta$ and $\theta_1, \ldots, \theta_n$ have the same effect on $Q_0$ and do not affect the variables in $V$, the resultant of $D'D$ is

$$Q_0^0 \theta' \leftarrow Q_n, Q' \theta.$$  

(*)

It follows from this that the resultant of $D'D$ is determined by the resultant of $D'$, by $G_0$, and by $R$.

(b): This part follows immediately from the formula (*). 0

We note for later use that since the resultant of an initial sequence of a derivation carries all the information needed to form subsequent resultants, if $D_1$
and $D_2$ have the same resultants, then identical continuations of the derivations $D'D_1$ and $D'D_2$ will also have the same resultant.

**Lemma 4.5.** The resultant of SLD-resolution on $\leftarrow A$ using the clause $A\theta \leftarrow Q$ is $A\theta \leftarrow Q$. So, if the clause $C$ is the resultant of an SLDNF-derivation from $\leftarrow A$, then the resultant of SLD-resolution on $\leftarrow A$ using $C$ is $C$.

**Proof.** If $\alpha$ is the renaming substitution and $\theta'$ the mgu used, then the resultant $R$ is $(A\theta \leftarrow Q)\alpha \theta'$. But $A\theta \leftarrow Q$ is obtainable by GSLD-resolution from $\leftarrow A$ using $A\theta \leftarrow Q$. Hence, by the lifting lemma, $(A\theta \leftarrow Q)\alpha \theta' = R\gamma$ for some $\gamma$. Thus $R$ is a variant of $A\theta \leftarrow Q$. The last part follows because the resultant of a derivation from $\leftarrow A$ is of the form $A\theta \leftarrow Q$. □

**Lemma 4.6 (Switching).** If an SLDNF-derivation selects at some step the literal $L_1$ and then the literal into which $L_1$ is sent, then the same resultant can be obtained by selecting $L_2$ first and then the literal into which $L_1$ is sent, provided $L_2$ is not a nonground negative literal.

**Proof.** If either $L_1$ or $L_2$ is a ground negative literal and a negation as failure step is used, the result is obvious, since the rest of the goal is left unchanged, and a resolution step leaves a ground negative literal unchanged. Thus we can suppose the steps in which $L_1$ and $L_2$ are selected are both resolution steps.

By the subderivation lemma, it is enough to consider a goal $\leftarrow L_1, L_2$. If the first two steps use $C_1: A_1 \leftarrow Q_1$ with mgu $\theta_1$, followed by $C_2: A_2 \leftarrow Q_2$ with mgu $\theta_2$, then $L_1\theta_1 = A_1\theta_1$ and $L_2\theta_1 \theta_2 = A_2\theta_2$, and the resultant is $(L_1, L_2)\theta_1 \theta_2 \leftarrow (Q_1\theta_1, Q_2)\theta_2$. This can be obtained by GSLD-resolution using first $C_2$ with goal substitution $\theta_1 \theta_2$ and clause substitution $\theta_2$ to obtain the resultant $(L_1, L_2)\theta_1 \theta_2 \leftarrow L_1\theta_1 \theta_2, Q_2\theta_2$, then using $C_1$ with the identity $\epsilon$ as the goal substitution and $\theta_1 \theta_2$ as the clause substitution, giving the same resultant as before. By the lifting lemma, this can also be obtained by SLD-derivation using $C_2$ and $C_1$, followed by a substitution. By symmetry, this substitution can be taken to be a renaming substitution. □

**Lemma 4.7 (Subtree).** Let $P$ be a normal program, $G$ a normal goal, and $\theta$ a substitution. If $T$ is an SLDNF-tree for $P \cup \{G\}$, then there is an SLDNF-tree $T'$ for $P \cup \{G\}\theta$, which is obtained by applying substitutions to the goals of a subtree of $T$ and which has the property that, for any goal $H$ in $T$ which has a corresponding goal $H\phi$ in $T'$, if $L$ is selected in $H$, then the corresponding literal $L\phi$ is selected in $H\phi$.

**Proof.** Since $T$ may be infinite, we prove by induction on $n$ that this property holds for the tree truncated at depth $n$. The case $n = 1$ is obvious. For the induction step, we want to show that, if there is a derivation step on a goal $G'\theta'$ using the clause $C$ giving a new goal $G_1$, then there is a derivation step on $G'$ using clause $C$ giving new goal $G'_1$ such that $G_1 = G'_1 \gamma$, for some $\gamma$. In fact, this follows from the lifting lemma. When the derivation step on the goal $G'$ in $T$ is a negation as failure step, the same step can be applied to $G'\theta'$. □
Note that there may also be SLDNF-trees for \( G\theta \) which are not subtrees of SLDNF-trees for \( G \), if negation as failure steps are used, because negative literals arising from \( G\theta \) may be ground, whereas those arising from \( G \) may not.

**Lemma 4.8.** Let \( P \) be a normal program, \( G \) a normal goal, and \( \theta \) a substitution. If \( P \cup \{G\} \) has a finitely failed SLDNF-tree of height \( h \), then \( P \cup \{G\theta\} \) has one of height \( \leq h \).

**Proof.** The lemma is an immediate consequence of the subtree lemma. \( \square \)

Note that there is an SLD version of Lemma 4.8 in which each occurrence of “SLDNF-tree” in the statement is replaced by “SLD-tree”.

An SLDNF-derivation \( G_0, \ldots, G_n \) is the initial part of a branch of an SLDNF-tree. The proof of the subtree lemma shows that either there is a corresponding derivation \( G'_0, \ldots, G'_n \) from \( G' = G_0\theta \) with each \( G'_i = G_i\gamma_i \), using the same clauses and selecting the literal \( L\gamma_i \) at the \( i \)th step when the original selects \( L \), or else this new derivation terminates without resultant at the \( i \)th step because \( L\gamma_i \) doesn't unify with the head of the clause with which \( L \) unifies.

**Lemma 4.9.** Let \( R \) be the resultant of an SLDNF-derivation \( D \) from a normal goal \( \leftarrow Q \), and \( \alpha \) a substitution. If there is a corresponding derivation \( D' \) from \( \leftarrow Q\alpha \), then its resultant \( R' \) is an instance of \( R \).

**Proof.** We suppose at first that \( D \) is an SLD-derivation. Let \( D = \leftarrow Q_0, \ldots, \leftarrow Q_n, \) where \( Q_0 = Q \). Let \( \theta_1, \ldots, \theta_n \) be the mgu used, and \( R_0, \ldots, R_n \) the resultants after steps 0, \ldots, \( n \). Similarly for \( D' \), let \( \theta'_1, \ldots, \theta'_n \) be the mgu used, and \( R'_0, \ldots, R'_n \) the resultants after steps 0, \ldots, \( n \). We prove by induction on \( n \) that there exists \( \alpha_n \) such that \( R'_n = R_n\alpha_n \). If \( n = 0 \), then \( R_0 = Q \leftarrow Q \) and \( R'_0 = Q\alpha \leftarrow Q\alpha \), so this is true with \( \alpha_0 = \alpha \).

For the induction step, we have that \( R_n = Q\theta_1 \ldots \theta_n \leftarrow Q_n \) and \( R'_n = R_n\alpha_n \). Then \( R'_{n+1} \) is the same as the resultant of a GSLD-derivation \( \leftarrow Q_0, \ldots, \leftarrow Q_{n+1} \), where the last step is a GSLD-step with goal substitution \( \alpha_n\theta'_{n+1} \) and clause substitution \( \theta'_{n+1} \). By the lifting lemma, \( R'_{n+1} \) is an instance of \( R_{n+1} \).

Now suppose that \( D \) has negation as failure steps. From \( D \) we can construct an SLD-derivation \( D_1 \) which is the same as \( D \), except that the negation as failure steps of \( D \) are not carried out. By the first part of the proof, the SLD-derivation \( D'_1 \) from \( \leftarrow Q\alpha \) corresponding to \( D_1 \) has a resultant which is an instance of the resultant for \( D_1 \). We can now carry out the corresponding negation as failure steps in \( D_1 \) and \( D'_1 \) to obtain the result. \( \square \)

**Lemma 4.10 (Persistence of failure).** Let \( P \) be a normal program and \( G \) a normal goal. If \( P \cup \{G\} \) has a finitely failed SLDNF-tree of height \( h \) and there is an SLDNF-derivation from \( G \) to \( G_1 \), then \( P \cup \{G_1\} \) has a finitely failed SLDNF-tree of height \( \leq h \).

**Proof.** Clearly it is enough to prove this for a derivation from \( G \) to \( G_1 \) of length one. Let \( L_1 \) be the literal of \( G \) selected in the first step of the finitely failed tree, and \( L_2 \) the literal selected in the first step of the derivation from \( G \) to \( G_1 \). If \( L_1 = L_2 \), then the result is obvious, indeed, with \( < h \) instead of \( \leq h \). If \( L_1 \neq L_2 \),
we prove the result by induction on \(h\). If \(h = 1\), then the first step of the derivation replaces \(L_1\) with \(L_1\theta\) for some \(\theta\). If \(L_1\) is positive and does not unify with the head of any clause of \(P\), then neither does \(L_1\theta\). If \(L_1\) is \(-A\), where \(A\) is ground and \(P \cup \{\leftarrow A\}\) has an SLDNF-refutation, then \(L_1\theta = L_1\) and so the same applies to it.

For height \(h + 1\), by the induction hypothesis, if we select first \(L_1\) and next (the resulting instance of) \(L_2\), then all the grandchildren of \(G\) have finitely failed trees of height \(\leq h\). If either of \(L_1\) or \(L_2\) is negative, it must be ground because it is selected. Hence we may apply the switching lemma to show that these grandchildren are the same as those obtained by first selecting \(L_2\) and next (the resulting instance of) \(L_1\). So their fathers, i.e., the children of \(G\) when \(L_2\) is selected first, have finitely failed trees of height \(\leq h + 1\). □

Note that there is an SLD version of the persistence of failure lemma in which each occurrence of “SLDNF” in the statement is replaced by “SLD”.

The next two lemmas establish the crucial relationships between following a derivation and using the single resultant clause corresponding to that derivation.

**Lemma 4.11**

(a) Let \(R\) be the resultant of an SLD-derivation \(D\) from \(\leftarrow A\). Let \(A'\) be an atom which unifies with the head of \(R\), and \(R''\) the resultant of the one step derivation from \(\leftarrow A'\) using the clause \(R\). Then there is a derivation \(D'\) from \(\leftarrow A'\) corresponding to \(D\) which selects at each step a corresponding literal and uses (a variant of) the same clause as \(D\) such that \(R''\) is an instance of the resultant \(R'\) of \(D'\).

(b) Any use of \(R\) in an SLD-derivation can be replaced by use of a derivation corresponding to \(D\). When this is done the resultant of the original derivation is an instance of the resultant of the new one.

**Proof.** (a): The proof is by induction on the length \(n\) of the derivation \(D\) from \(\leftarrow A\). If \(n = 0\), then \(R\) is \(A\leftarrow A\). Let \(R\alpha\) be a variant of \(R\) with no variables in common with \(A'\). If \(\theta\) is the mgu of \(A'\) and \(A\alpha\), then the resultant \(R''\) is \(A'\theta \leftarrow A'\theta\). Hence \(R''\) is an instance of \(A'\leftarrow A'\), which is the resultant \(R'\) of a derivation of length 0 from \(\leftarrow A'\).

For the induction step, let the resultant \(R_n\) of the first \(n\) derivation steps from \(\leftarrow A\) be \(A\theta \leftarrow Q_1, B, Q_2\), where \(B\) is the next selected atom. Suppose the \((n + 1)\)th step uses the clause \(C_{n+1}: B_1 \leftarrow Q'\) with mgu \(\theta'\). So the final resultant \(R\) is

\[
A\theta\theta' \leftarrow (Q_1, Q', Q_2)\theta'.
\]

The resultant \(R''\) of using \(R\) on \(\leftarrow A'\) is

\[
A\theta\theta'\alpha\theta'' \leftarrow (Q_1, Q', Q_2)\theta'\alpha\theta'',
\]

where \(\alpha\) is a renaming substitution for \(R\), and \(\theta''\) is an mgu of \(A'\) and \(A\theta\theta'\alpha\). Now it is easy to check that \(R''\) can be obtained by a GSLD-derivation from \(\leftarrow A'\) using \(R_n\) and \(C_{n+1}\). Hence, by the induction hypothesis and the lifting lemma, \(R''\) is an instance of the resultant \(R'\) of the corresponding SLD-derivation from \(\leftarrow A'\).

(b): This follows immediately from part (a), the subderivation lemma, and the lifting lemma. □
Lemma 4.12. Let $R$ be the resultant of an SLDNF-derivation from $\leftarrow A$, and $R''$ the resultant of using the clause $R$ on a goal of the form $\leftarrow A\phi$. Then there is a corresponding SLDNF-derivation from $\leftarrow A\phi$, and its resultant is $R''$.

**Proof.** The proof is by induction on the number $k$ of negation as failure steps in the SLDNF-derivation of length $m$ from $\leftarrow A$. If $k = 0$, Lemma 4.11 shows that the corresponding derivation from $\leftarrow A\phi$ exists. Let $R'$ be its resultant. By Lemma 4.11, $R''$ is an instance of $R'$. By Lemma 4.9, $R'$ is an instance of $R$. By Lemma 4.5, $R'$ is the resultant of using $R'$ on $\leftarrow A\phi$. Hence, by the lifting lemma, $R'$ is an instance of $R''$.

For the induction step, let the last negation as failure step be the $(n + 1)$th, and let

$$A\theta \leftarrow Q_1, \rightarrow B, Q_2$$

be the resultant $R_n$ of the first $n$ steps in the given derivation from $\leftarrow A$. We are supposing that $B$ is ground and $\leftarrow B$ fails finitely, so the next resultant $R_{n+1}$ is

$$A\theta \leftarrow Q_1, Q_2.$$

Since the head of $R_n$ unifies with $A\phi$, by the induction hypothesis, the resultant of the corresponding first $n$ steps from $\leftarrow A\phi$ exists and is the same as the resultant of $R_n$ on $\leftarrow A\phi$. We may suppose that $R_n$ is a variant with no variables in common with $A\phi$, so this resultant is

$$A\theta\theta' \leftarrow (Q_1, \rightarrow B, Q_2)\theta',$$

where $\theta'$ is an mgu of $A\phi$ and $A\theta$. Since $B$ is ground, the corresponding negation as failure step is applicable, giving for the resultant of the first $n + 1$ steps from $\leftarrow A\theta$

$$A\theta\theta' \leftarrow (Q_1, Q_2)\theta',$$

which is the same as the resultant of using $R_{n+1}$ on $\leftarrow A\phi$. Now the resultant of the whole derivation from $\leftarrow A$ is the same as the resultant of using $R_{n+1}$, followed by the $(n + 2)$th, ..., $m$th steps. Since these are all SLD-resolution steps, the case $k = 0$ shows that the resultant of using $R_m$ on $\leftarrow A\phi$ is the same as the resultant of using $R_{n+1}$, followed by the remaining steps. By what we have just proved, this is the same as the resultant of the whole corresponding derivation from $\leftarrow A\phi$. \qed

Note that not all SLDNF-derivations from $\leftarrow A\phi$ correspond in this way to SLDNF-derivations from $\leftarrow A$, for they may contain negation as failure steps which are not possible if we start from the goal $\leftarrow A$.

### 4.2. Partial Evaluation Theorems

We are now ready to present the partial evaluation theorems.

**Theorem 4.1.** Let $P$ be a normal program, $G$ a normal goal, $A$ a finite set of atoms, and $P'$ a partial evaluation of $P$ wrt $A$ using SLD-trees. Then the following hold.

(a)(i) If $P' \cup \{G\}$ has an SLD-refutation with computed answer $\theta$, then $P \cup \{G\}$ has an SLD-refutation with computed answer including $\theta$.

(ii) If $P \cup \{G\}$ has a finitely failed SLD-tree, then so does $P' \cup \{G\}$. 
PARTIAL EVALUATION IN LOGIC PROGRAMMING

(b)(i) Let $P' \cup \{G\}$ be $A$-closed. If $P \cup \{G\}$ has an SLD-refutation with computed answer $\theta$, then so does $P' \cup \{G\}$.

(ii) Let $P' \cup \{G\}$ be $A$-closed. If $P' \cup \{G\}$ has a finitely failed SLD-tree, then so does $P \cup \{G\}$.

PROOF. (a)(i): By Lemma 4.11, a refutation of $P' \cup \{G\}$ with computed answer $\theta$ can be expanded into a refutation of $P \cup \{G\}$ with answer including $\theta$.

(a)(ii): We prove by induction on $h$ that if there is a finitely failed SLD-tree $T$ for $P \cup \{G\}$ of height $h$, then there is a finitely failed SLD-tree for $P' \cup \{G\}$. If $h = 1$, then the selected atom of $G$ doesn’t match the head of any clause of $P$, so it doesn’t match the head of any clause of $P'$. Thus $T$ is also a finitely failed SLD-tree for $P' \cup \{G\}$. If $T$ has height $h + 1$ and the first selected atom $A$ does not contain any of the predicate symbols in $A$, then the first step is the same wrt $P'$. Hence the result follows by the induction hypothesis. Otherwise, to get an SLD-tree wrt $P'$, we must lead an edge out of the root goal $G$ for each of the clauses in $P'$ whose head matches $A$. By Lemma 4.11, the resultant of using such a clause $R$ is an instance of the resultant of using the expanded sequence of derivation steps corresponding to it. Now the first step of this takes us into one of the goals of the next level of $T$, which have finitely failed trees wrt $P$ of height $\leq h$. By (the SLD version of) the persistence of failure lemma, so does the goal $G'$ reached after the whole sequence of derivation steps. By (the SLD version of) Lemma 4.8, so does the goal $G' \theta$ reached by using $R$. Hence, by the induction hypothesis, it has a finitely failed tree wrt $P'$. This completes the induction step, since it shows that each of the next level goals of the tree wrt $P'$ has a finitely failed tree wrt $P'$.

(b)(i): The proof is by induction on the length $n$ of the refutation for $P \cup \{G\}$. If $n = 0$, then the result is obvious. The induction step is also obvious if the predicate symbol in the first selected atom $A$ is not one of the partially evaluated predicates. Otherwise, if $A$ contains a partially evaluated predicate, then, because $P' \cup \{G\}$ is $A$-closed, there is an atom $A_1 \in A$ and a substitution $\phi$ such that $A = A_1 \phi$. Now, since a refutation ends in the empty goal, all atoms are eventually chosen. Hence, we may use the switching lemma to put this refutation into the form where it starts off by following the “same” choice of atoms and clauses as some branch of the tree used in the partial evaluation of $A_1$. This new refutation must eventually reach a goal $G'_i$ corresponding to a goal $G_i$ in the set $\{G_1, \ldots, G_r\}$ of goals chosen in the partial evaluation of $A_1$. Let $R_i$ in $P'$ be the resultant corresponding to $G_j$. Clearly, by Lemma 4.9, the head of $R_i$ when standardized apart unifies with $A_1 \phi$. Hence $R_i$ can be used on the goal $\leftarrow A_1 \phi$, and, by Lemma 4.12, the resultant of doing this is the same as the resultant of the corresponding derivation from $\leftarrow A_1 \phi$. By the subderivation lemma, it follows that this first part of the new refutation for $G$ down to $G'_i$ can be obtained by using the clause $R_i$. Now we apply the induction hypothesis and the subderivation lemma.

(b)(ii): The proof is by induction on the height of the finitely failed tree $T'$ for $P' \cup \{G\}$. If this height is 1, then the selected atom $A$ in $G$ doesn’t match the head of any clause of $P'$. If the predicate symbol in $A$ is not one of the partially evaluated predicates, then it doesn’t match the head of any clause of $P$ either. If $A$ does contain a partially evaluated predicate, then by the closedness condition, it is of the form $A_1 \phi$ for an atom $A_1 \in A$. By the subtree lemma, there is a tree wrt
$P$ for $\leftarrow A_1\phi$ which is an instance of a subtree of the tree used in the partial evaluation of $A_1$. This must be finitely failed; otherwise, it reaches one of the goals $G_1\theta_1, \ldots, G_r\theta_r$, say $G_i\theta_i$, with a resultant which is an instance of the resultant $R_i$ corresponding to $G_i$. Hence the head of $R_i$ when standardized apart unifies with $A_1\phi$, which is a contradiction. Clearly we can now obtain a finitely failed SLD-tree for $P \cup \{G\}$.

The induction step is similar. In that case, each of the goals $G_i\theta_i$ reached is, by Lemma 4.12, the same as the one obtained by using $R_i$ on $A_1\phi$. So each is contained in a child of $G$ in the tree $T'$. Using the induction hypothesis, we now obtain a finitely failed SLD-tree for $P \cup \{G\}$. □

The condition in the definition of partial evaluation that the goals $G_1, \ldots, G_r$ are nonroot goals is used in the proofs of parts (a)(ii) and (b)(i) of Theorem 4.1. The following examples show that this condition cannot be dropped.

**Example.** Let $P$ be the definite program

$$p \leftarrow q$$

and $A$ be $\{q\}$. Then the definite program $P'$

$$p \leftarrow p$$

would be a partial evaluation of $P$ wrt $A$ if the nonroot condition were dropped. However, $P \cup \{\leftarrow p\}$ has a finitely failed SLD-tree, but $P' \cup \{\leftarrow p\}$ does not.

**Example.** Let $P$ be the definite program

$$p \leftarrow$$

and $A$ be $\{p\}$. Then the definite program $P'$

$$p \leftarrow p$$

would be a partial evaluation of $P$ wrt $A$ if the nonroot condition were dropped. However, $P \cup \{\leftarrow p\}$ has an SLD-refutation, but $P' \cup \{\leftarrow p\}$ does not.

We have given the above version 4.1 of the partial evaluation theorem first because it shows clearly the relationship between the computational strengths of $P$ and $P'$. For successful goals, $P'$ is in general weaker than $P$: if a goal succeeds under $P'$, it succeeds under $P$. As usual, this relationship is reversed for failure: if a goal fails under $P$, it fails under $P'$. Since $P'$ was constructed under the implicit assumption that the only atoms containing the partially evaluated predicates which would involved in dealing with the goal were instances of ones in the set $A$, the desired equivalence between $P$ and $P'$ can only be expected for goals satisfying this assumption. The closedness condition of (b)(i) and (b)(ii) is a natural way of expressing this assumption.

By adding the condition that $A$ is independent, we obtain another version of the theorem as follows.

**Theorem 4.2.** Let $P$ be a normal program, $G$ a normal goal, $A$ a finite, independent set of atoms, and $P'$ a partial evaluation of $P$ wrt $A$ using SLD-trees such that
PARTIAL EVALUATION IN LOGIC PROGRAMMING

$P' \cup \{G\}$ is $A$-closed. Then the following hold:

(i) $P' \cup \{G\}$ has an SLD-refutation with computed answer $\theta$ iff $P \cup \{G\}$ does.
(ii) $P' \cup \{G\}$ has a finitely failed SLD-tree iff $P \cup \{G\}$ does.

The improvement in (i) to “computed answer $\theta$” instead of “computed answer including $\theta$” is obtained by using the closedness condition and the independence of $A$, and replacing the use of Lemma 4.11 with the use of Lemma 4.12 and the subderivation lemma.

A special case of Theorem 4.2(i) was given in [6]. Another result related to Theorem 4.2(i) was given in [18].

Note that the independence condition is only used in the “only if” half of Theorem 4.2(i). The following example shows that this condition cannot be dropped.

**Example.** Let $P$ be the definite program

$p(x) \leftarrow$, $A \upto \{p(x), p(a)\}$, and $P'$

$p(x) \leftarrow$

$p(a) \leftarrow$

a partial evaluation of $P$ wrt $A$. Then $P' \cup \{p(x)\}$ is $A$-closed and also has a refutation with computed answer $\{x/a\}$. However, while $P \cup \{p(x)\}$ does have a refutation, it does not have one with computed answer $\{x/a\}$.

**Theorem 4.3.** Let $P$ be a normal program, $G$ a normal goal, $A$ a finite, independent set of atoms, and $P'$ a partial evaluation of $P$ wrt $A$ such that $P' \cup \{G\}$ is $A$-closed. Then the following hold:

(i) $P' \cup \{G\}$ has an SLDNF-refutation with computed answer $\theta$ iff $P \cup \{G\}$ does.
(ii) $P' \cup \{G\}$ has a finitely failed SLDNF-tree iff $P \cup \{G\}$ does.

**Proof.** The proof is by induction on the rank (depth of nesting of negation as failure calls [20]). We shall prove (i) [(ii)] for a given SLDNF-refutation [finitely failed SLDNF-tree] of rank $k$, on the assumption that both (i) and (ii) hold for lesser ranks.

(i): Suppose we have an SLDNF-refutation of $P' \cup \{G\}$ of rank $k$. Each application of one of the new clauses of $P'$ must be on a selected atom which is an instance of one of the atoms in $A$. By the subderivation lemma, Lemma 4.12, the closedness condition, and the independence of $A$, the application can be expanded into a derivation wrt $P$. A negation as failure step takes the form of deleting a literal $\neg A$, where $P' \cup \{\neg A\}$ has a finitely failed tree of rank $< k$. Since $\{A\}$ is $A$-closed, the induction hypothesis shows that $P \cup \{\neg A\}$ has a finitely failed tree, so that this negation as failure step is legitimate wrt $P$ and so the whole derivation becomes a refutation of $P \cup \{G\}$ with the same computed answer.
Conversely, suppose that we have an SLDNF-refutation of $P \cup \{G\}$ of rank $k$. The proof will be by induction on the length of this and is similar to the proof of Theorem 4.1(b)(i), except when the first step is a negation as failure step with selected literal $\neg A$, where $A$ is ground and $P \cup \{\neg A\}$ has a finitely failed tree of rank $< k$. Since $\{A\}$ is A-closed, by the induction hypothesis on rank, $P' \cup \{\neg A\}$ has a finitely failed tree. So we can make the same negation as failure step wrt $P'$. The new goal is A-closed, so we can now complete the argument by using the induction hypothesis on length.

(ii): Suppose that we have a finitely failed tree $T'$ of rank $k$ for $P' \cup \{G\}$. We prove by induction on the height of $T'$ that there is one for $P \cup \{G\}$. The argument is similar to the proof of Theorem 4.1(b)(ii), except that there are now negation as failure steps to consider. For height 1, we may have selected a ground negative literal $\neg A$, where $P' \cup \{\neg A\}$ has a refutation of rank $< k$. Since $\{A\}$ is A-closed, the induction hypothesis on rank gives a refutation of $P \cup \{\neg A\}$. Hence $T'$ is a finitely failed tree of height 1 for $P \cup \{G\}$. There is also a new case in the induction on height, where a ground negative literal $\neg A$ is selected and $P' \cup \{\neg A\}$ has a finitely failed tree of rank $< k$. Again, the induction hypothesis on rank gives a finitely failed tree for $P \cup \{\neg A\}$.

Conversely, if $T$ is a finitely failed tree of rank $k$ for $P \cup \{G\}$, we prove by induction on its height that there is one for $P' \cup \{G\}$. The proof is similar to the proof of Theorem 4.1(a)(ii), except for the negation as failure steps and the need to check closedness. There is a new case for height 1, where the selected literal is a ground negative literal $\neg A$ and $P \cup \{\neg A\}$ has a refutation of rank $< k$. Since $\{A\}$ is A-closed, the induction hypothesis on rank gives a refutation of $P' \cup \{\neg A\}$. There is a new case in the induction step, where the selected literal is a ground negative literal $\neg A$ and $P \cup \{\neg A\}$ has a finitely failed tree of rank $< k$. Since $\{A\}$ is A-closed, the induction hypothesis on rank gives a finitely failed tree of rank $< k$ for $P' \cup \{\neg A\}$. The induction hypothesis on height now completes the argument as before. When the selected literal is a positive literal containing one of the partially evaluated predicates, we proceed as in the proof of Theorem 4.1(a)(ii), but replacing the use of Lemma 4.11 with the use of Lemma 4.12, the subderivation lemma, and the independence of $A$. □

The independence condition is only used in the "only if" half of Theorem 4.3(i) and the "if" half of Theorem 4.3(ii). The example after Theorem 4.2 also shows that the independence condition cannot be dropped from Theorem 4.3. Furthermore, the following example shows that if the independence condition is dropped, then the computation of $P \cup \{G\}$ in Theorem 4.3(i) can flounder.

Example. Let $P$ be the normal program
\[
p(x) \leftarrow \neg q(x),
\]
A the set $\{p(x), p(a)\}$, and $P'$
\[
p(x) \leftarrow \neg q(x)
\]
\[
p(a) \leftarrow
\]
a partial evaluation of $P$ wrt A. Then $P' \cup \{\neg p(x)\}$ is A-closed and also has a
PARTIAL EVALUATION IN LOGIC PROGRAMMING

refutation with computed answer \( \{x/a\} \). However, the computation of \( P \cup \{ \leftarrow p(x) \} \) flounders.

The condition in Theorem 4.3 that \( P' \cup \{G\} \) is \( A \)-closed can be replaced by a weaker condition.

**Definition.** Let \( P \) be a normal program and \( G \) a normal goal. We say \( G \) depends upon a predicate \( p \) in \( P \) if there is a path from a predicate in \( G \) to \( p \) in the dependency graph for \( P \).

**Definition.** Let \( P \) be a normal program, \( G \) a normal goal, \( A \) a finite set of atoms, \( P' \) a partial evaluation of \( P \) wrt \( A \), and \( P^* \) the subprogram of \( P' \) consisting of the definitions of predicates in \( P' \) upon which \( G \) depends. We say \( P' \cup \{G\} \) is \( A \)-covered if \( P^* \cup \{G\} \) is \( A \)-closed.

Theorem 4.3 can then be strengthened by replacing the condition that \( P' \cup \{G\} \) is \( A \)-closed with the condition that \( P' \cup \{G\} \) is \( A \)-covered. The proof of the strengthened result requires only very minor and obvious changes to that of Theorem 4.3. A partial evaluation procedure based on the strengthened form of Theorem 4.3 is given in [2].

Partial evaluation as defined here is a combination of "specialization" and "unfolding". An interesting special case is where there is no specialization, just unfolding of a single clause. In this case, the closedness and independence conditions are automatically satisfied, so the program \( P' \) obtained from \( P \) by replacing a clause \( C \) in \( P \) with a set of unfoldings of \( C \) is fully computationally equivalent to \( P \). To be precise, let \( C \) be \( A \leftarrow Q \). By an unfolding of \( C \), we mean a set of resultants of the form

\[ A \theta_i \leftarrow Q_i \]

corresponding to a choice of goals \( \leftarrow Q_1, \ldots, \leftarrow Q_r \), one of each nonfailing branch of an SLDNF-tree for \( \leftarrow Q \), where \( \theta_i \) is the computed answer for the derivation from \( \leftarrow Q \) down to \( \leftarrow Q_i \) (i = 1, \ldots, r). The resulting program \( P' \) can be obtained as a partial evaluation of \( P \) in our sense by taking the set \( A \) of atoms to be the singleton \( \{ p(x_1, \ldots, x_n) \} \) of the most general atom containing the predicate \( p \) occurring in \( A \). We then take a tree for \( \leftarrow p(x_1, \ldots, x_n) \), stop at the end of the first edge for all clauses except \( C \), and continue the branch using \( C \) to the end goals \( \leftarrow Q_1, \ldots, \leftarrow Q_r \).

4.3. Dependence on the Computation Rule; Results for PROLOG

We now discuss the effect on the results when the partial evaluations and the refutations (or the finitely failed trees) are all produced using (safe) computation rules. Note that we use "computation rule" in the sense of [24] rather than the narrower sense of [20], where a computation rule is a function of the goal alone. Theorems 4.1, 4.2, and 4.3 have all been proved, in particular, for all computation rules used in obtaining the partial evaluation \( P' \) of \( P \). However, as far as the goal \( G \) is concerned, they take the form, for example, "if \( P' \cup \{G\} \) has a refutation using some computation rule, then \( P \cup \{G\} \) has a refutation, which may use
another, possibly different, computation rule”. The results hold in some cases when the same fixed rule is used throughout, e.g., for the maximal rules shown to exist in [24]. (These may not be of much practical interest, because there may be no recursive maximal rule [25]). As the following example shows, they do not hold in all cases when the same fixed rule is used throughout.

Let us make the PROLOG “first literal” computation rule safe by taking it to be:

“take the first possible (i.e., positive or ground negative) literal”

and use this rule throughout. Let the program $P$ be

$$
\begin{align*}
  r(x) &\leftarrow r(x) \\
  q(x) &\leftarrow \neg r(x), p(x) \\
  s &\leftarrow \neg q(a).
\end{align*}
$$

By partially evaluating $P$ wrt $\{q(x)\}$, we can obtain the program $P'$, which is the same as $P$ except that the second clause is deleted. Now $P' \cup \{\leftarrow s\}$ has an SLDNF-refutation, but in $P$ the derivation from $\leftarrow s$ comes to a dead end.

Not surprisingly, the results don't hold for the unsafe PROLOG first literal rule with negation as failure applied to nonground negative literals. For example, let $P$ be

$$
\begin{align*}
  p(a) &\leftarrow q(b) \\
  q(x) &\leftarrow \neg r(x) \\
  r(a) &\leftarrow
\end{align*}
$$

By partially evaluating $P$ wrt $\{q(x)\}$, we can obtain the program $P'$, which is the same as $P$ except that the second clause is deleted. Now $\leftarrow p(a)$ succeeds in $P$, but fails in $P'$.

When $P$ is a definite program and $G$ is a definite goal, there are better results. Since all computation rules are equally powerful for SLD-resolution as far as success is concerned, parts (a)(i) and (b)(i) (concerning refutations) of Theorem 4.1 can be stated for any three computation rules, one to be used in the partial evaluation, another for $P$, and a third for $P'$. This is not true for the failure results (a)(ii) and (b)(ii), even if the same rule is used throughout. Consider the program $P$

$$
\begin{align*}
  p &\leftarrow p, q
\end{align*}
$$

with the computation rule

“take the second literal (if there is more than one) ”.

Using this rule, the empty program $P'$ is a partial evaluation of $P$ wrt $\{p\}$. The goal $\leftarrow p, p$ fails wrt $P'$, using this rule, but gives an infinite derivation wrt $P$. However, if the PROLOG first literal rule is used throughout, our proof of these failure results remains valid. The failure results are also valid for any fair rule, for example, the rule which cycles literals around (last to first) and then takes the first literal. This follows from the completeness of negation as failure for definite programs [20, Theorem 16.1].

When the depth-first search rule of PROLOG is taken into account, the order in which we write down the resultants in the partially evaluated program becomes
important in maintaining the computational equivalence between the original program and the partially evaluated program. Resultants should be ordered according to the lexicographic order of the clause sequences \(C_i, \ldots, C_n\) used in their derivations. For example, if \(P\) is

\[
\begin{align*}
p(x, y) &\leftarrow q(x, y) \\
q(f(x), y) &\leftarrow q(f(x), y) \\
q(x, g(y)) &\leftarrow
\end{align*}
\]

and we partially evaluate \(P\) wrt \(\{p(f(x), y), p(x, g(y))\}\) to obtain \(P'\), we must write \(P'\) as

\[
\begin{align*}
p(f(x), y) &\leftarrow q(f(x), y) \\
p(f(x), g(y)) &\leftarrow q(f(x), g(y)) \\
p(x, g(y)) &\leftarrow \\
q(f(x), y) &\leftarrow q(f(x), y) \\
q(x, g(y)) &\leftarrow
\end{align*}
\]

but not as

\[
\begin{align*}
p(x, g(y)) &\leftarrow \\
p(f(x), y) &\leftarrow q(f(x), y) \\
p(f(x), g(y)) &\leftarrow q(f(x), g(y)) \\
q(f(x), y) &\leftarrow q(f(x), y) \\
q(x, g(y)) &\leftarrow
\end{align*}
\]

for the last program makes \(\leftarrow p(f(x), g(y))\) succeed, whereas it loops on the original program.

5. CONCLUSION

In this paper, we have proved theorems which provide a foundation for partial evaluation in logic programming. We have identified the closedness and independence conditions which are needed to ensure the soundness and completeness for the procedural semantics of the partial evaluation process. Thus a practical consequence of our results is that partial evaluators need to enforce these conditions.

However, there are other issues which need to be resolved before partial evaluation can realize its full potential. One of the most difficult of these is the control of the partial evaluation, that is, knowing when to unfold and when not to unfold. The issue is, of course, very closely related to the problems of loop trapping and providing coroutining control in PROLOG systems, and previous work on both these topics can be expected to be useful here. Most current partial evaluators use rather crude stopping conditions. For example, the stopping condition “don’t unfold the selected literal if it occurred earlier in the derivation” (and variations on this) is commonly used. This kind of control is not sophisticated enough for many applications, especially as we want the partial evaluators to be as
autonomous as possible. More details on flexible computation rules for partial evaluation can be found in [2].

Another problem is the increase in the size of the code of the partially evaluated program. In general terms, there is a tradeoff between the depth to which partial evaluation can be carried and the size of the corresponding partially evaluated code. For certain types of programs, a naive partial evaluator can easily produce an exponential explosion in the amount of such code. The research problem here is to build partial evaluators which have sufficiently sophisticated knowledge about partial evaluation in general and about the particular program being partially evaluated in order to prune failing branches, for example, so as to keep the code size down to a reasonable level.

We have concentrated on partial evaluation for the "applications" family of logic programming languages, which includes the various dialects of PROLOG. Our results (particularly Theorems 4.1 and 4.2) are also applicable to the "systems" family, which includes Parlog, concurrent PROLOG, and GHC. However, a complete treatment of partial evaluation for the "systems" languages would require an investigation of the commit operators used in these languages. Generalizations of the results of this paper to programs with a pruning operator, which includes the commit operator, are contained in [30].

Finally, there is once again the problem of impure features of PROLOG. Our theory only applies directly to pure PROLOG programs, or at most those parts of a PROLOG program which are pure. As is pointed out in [29], for example, the impure features of PROLOG cause significant complications if one wants to obtain a partially evaluated program which is computationally equivalent to the original program. This situation emphasizes yet again the urgent need to eliminate (or at least narrow) the gap between the theory and the practice in logic programming. The impure features of PROLOG need to be cleaned up sufficiently so that (some extension of) the existing theory can be applied to them. Only then shall we be in a position to produce provably correct programs.

We thank Kerima Benkerimi for the suggestions that the independence condition replace a more restrictive condition used in an earlier version of this paper and that the condition that $P' \cup \{G\}$ is $A$-covered replace the condition that $P' \cup \{G\}$ is $A$-closed in Theorem 4.3.

REFERENCES


