A NOTE ON THE EXHAUSTIVENESS OF SLDNF-RESOLUTION FOR NORMAL PROGRAMS

Giuseppe De Giacomo

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italia
Email: degiacom@assi.ing.uniroma1.it

Abstract

We prove, by means of results on partial evaluation, that: given a normal program P, whose completion $comp(P)$ is consistent, and a normal goal G, if $P \cup \{G\}$ has a finite non-failing SLDNF-tree T, then the computed answers resulting from T are *all* the correct answers for $comp(P) \cup \{G\}$ (modulo instantiation).

1. Introduction

In this paper we present the following result: let P be a normal program such that $comp(P)$ is consistent, and let G be a normal goal; if $P \cup \{G\}$ has a finite non-failing tree T, then the answers computed by T are *all* the correct answers for $comp(P) \cup \{G\}$ (modulo instantiation).

A full proof of this claim (first stated by Clark in [2]), has never been published. We prove it here by using the machinery for *partial evaluation* developed by Lloyd and Shepherdson ([4]).

Such an exhaustiveness result has been somewhat disregarded in the classical literature on SLDNF-resolution. For example, Lloyd does not report it at all in [3]. Shepherdson just mentioned it in connection with completeness results for SLDNF-resolution, ([5] and [6]), pointing out that it is not very useful by itself because it is not easy to decide whether a goal has a finite SLDNF-tree (actually such a property is tightly bound with the termination problem for SLDNF-resolution, which is undecidable).

From a practical point of view the result has its own importance. In fact, as a consequence of it we have that every time an SLDNF-resolution interpreter (e.g. a sound Prolog) terminates while evaluating a goal, it provides a complete logical characterization of the goal. Indeed, if the goal fails, by Clark's theorem on soundness of negation as failure, the goal is false. On the other hand, if any answer is actually computed, by the result proved here, the computed answers are all the correct answers for it (modulo instantiation).

The the paper is organized as follows: after summarizing some well-known definitions of logic programming in Section 2, we introduce partial evaluation in Section 3, and finally we prove the result in Section 4.

2. Preliminaries

We assume the reader familiar with the standard theoretical results of logic programming, which are contained in [3]. The basic definitions are briefly recalled here.

Definition A *normal program* is a finite set of *program clauses* of the form

$$A \leftarrow L_1, \ldots, L_n \ ,$$

where A is an atom and $L_1, \ldots, L_n$ are literals. 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. A *normal goal* $\leftarrow W$ is a clause of the form

$$\leftarrow L_1, \ldots, L_n \ ,$$

where $L_1, \ldots, L_n$ are literals. □

As usual, the *completion* of a program P, denoted as $comp(P)$, is the collection of *completed definitions* of the predicate symbols in P together with Clark's Equality Theory.

Definition Let P be a normal program and $\leftarrow W$ a normal goal. A *correct answer* for $comp(P) \cup \{\leftarrow W\}$ is a substitution θ for the free variables in W such that $comp(P)$ implies the universal closure of $W\theta$:

$$comp(P) \models \forall W\theta.$$

□

3. On partial evaluation

In view of the literature on partial evaluation (cf. [4], [1]) it is convenient to use slightly more general definitions of SLDNF-derivation and SLDNF-tree here than those given in [3]. In [3], an SLDNF-derivation is either infinite, successful or failed. We also allow it to be *incomplete*, in the sense that at any step we are allowed to simply not select any literal and terminate the derivation. Likewise in SLDNF-tree we may neglect to unfold a goal.

A concept that is needed to define partial evaluation is that of *resultant*.

Definition A *resultant* is a first order formula of the form

$$Q_1 \leftarrow Q_2,$$

where Q_i $(i = 1, 2)$, either is missing, or (if present) is a conjunction of literals. Any variables in Q_1 or Q_2 are assumed to be universally quantified at the front of the resultant. □

Notice that, a resultant is not a clause, in general, because Q_1 stands for a conjunction and not a disjunction of literals.

Definition Let P be a normal program, G a normal goal $\leftarrow \ Q$, and $G_0 = G, G_1, \ldots, G_n$ an SLDNF-derivation $P \cup \{G\}$, where the sequence of substitutions is $\theta_1, \ldots, \theta_n$ and G_n is $\leftarrow Q_n$. Let θ 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* θ and *resultant* $Q\theta \leftarrow Q_n$. (Notice that, if $n = 0$, then the resultant is $Q \leftarrow Q$.) □

It can be shown that the resultant $Q\theta \leftarrow Q_n$ of an SLDNF-derivation from Q down to the goal Q_n, is a logical consequence of the completed definition of the predicate symbols in the heads of the (input) clauses used in the derivation together with the associated Clark's Equality Theory.

Now, we give the definition of *partial evaluation* (shortly *PE*). Note that the definition refers to three kinds of PE: the PE of an atom in a program, of a set of atoms in a program, and of a program wrt a set of atoms.

Definition Let P be a normal program, A an atom, and T a (not necessarily complete) SLDNF-tree for $P \cup \{\leftarrow A\}$. Let $G_1, \ldots, G_r$ be (non-root) goals in T chosen so that each non-failed 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 associated with the branch leading to G_i.

- The set of resultants $\pi = \{R_1, \ \ldots, R_r\}$ is a *PE of* A *in* P. These resultants have the following form:
$$R_i = A\theta_i \leftarrow Q_i \ \ (i = 1, \ldots, r).$$
where we have assumed $G_i = \leftarrow Q_i$

- Let $\mathbf{A} = \{A_1, \ldots, A_s\}$ be a finite set of atoms, and π_i $(i = 1, \ldots, s)$ a PE of A_i in P. Then $\Pi = \pi_1 \cup \ldots \cup \pi s$ is a *PE of* $\mathbf{A}$ *in* P.

- Let P' be the normal program resulting from P when the definitions therein of the predicate symbols in $\mathbf{A}$ are replaced by a PE of $\mathbf{A}$ in P. Then P' is a *PE of* P *wrt* $\mathbf{A}$.

□

The next theorem is the main result on the declarative semantics. First we report the definition of *closedness condition* to be used in the theorem.

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

The reason we need this condition is, intuitively, that if we "specialize" the definition of a predicate symbol p wrt an atom A containing p, then we can not expect to be able to correctly answer calls to p that are not instances of A.

Theorem 1 (Lloyd Shepherdson) *Let P be a normal program, W a closed first order formula, $\mathbf{A}$ a finite set of atoms, and P' a PE of P wrt $\mathbf{A}$ such that $P' \cup \{W\}$ is $\mathbf{A}$-closed. If W is a logical consequence of $comp(P')$, then is a logical consequence of $comp(P)$.*

The converse of this theorem does not hold, as the following classical example shows.

Example By partially evaluating wrt $A = \{r, s\}$ the following stratified normal program P

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

we can obtain the program P'

$$p \leftarrow \sim q$$
$$q \leftarrow r, \sim s$$
$$r \leftarrow r$$
$$s \leftarrow s.$$

Now p is a logical consequence of $comp(P)$, but not of $comp(P')$. □

4. An exhaustiveness theorem

Here we arrive at the core of this paper:

Theorem 2 *Let P be a normal program such that $comp(P)$ is consistent, and let G be a normal goal. If $P \cup \{G\}$ has a (complete) non-failing finite SLDNF-tree T, then for every correct answer θ for $comp(P) \cup \{G\}$ there exists an SLDNF-refutation in T with computed answer σ, and a substitution γ such that $\theta = \sigma\gamma$.*

Proof Let G be $\leftarrow W$, ans a predicate symbol not occurring in P defined as

$$ans(X) \leftarrow W,$$

where X are the free variables occurring in W, and P^{ans} is the normal program

$$P \cup \{ans(X) \leftarrow W\}.$$

Let T' be the SLDNF-tree obtained from T adding an arc leading from $\leftarrow ans(X)$ to $\leftarrow W$ (notice that, every SLDNF-tree for $P^{ans} \cup \{ans(X)\}$ has the goal $\leftarrow W$ at depth 1). Consider the following PE of ans in P^{ans} obtained from the non failing leaves of T'

$$ans(X)\theta_1 \leftarrow$$
$$\vdots$$
$$ans(X)\theta_r \leftarrow,$$

where $\theta_i = \{X/Ti(Y_i)\}$, $T_i(Y_i)$ are tuples of terms, and Y_i are the free variables therein. Let $P^{ans'}$ be the corresponding PE of P^{ans} wrt $\mathbf{A} = \{ans(X)\}$. The completed definition for ans in $P^{ans'}$ is

$$\forall X(ans(X) \leftrightarrow \exists Y_1(X = T_1) \vee \ldots \vee \exists Y_r(X = T_r)).$$

The above formula is **A**-closed, hence by Theorem 1 we have

$$comp(P^{ans}) \models \forall X(ans(X) \leftrightarrow \exists Y_1(X = T_1) \vee \ldots \vee \exists Y_r(X = T_r)).$$

Considering the completed definition for ans in P^{ans}, i.e. $\forall X(ans(X) \leftrightarrow W)$, we can replace $ans(X)$ by W, getting

$$comp(P^{ans}) \models \forall X(W \leftrightarrow \exists Y_1(X = T_1) \vee \ldots \vee \exists Y_r(X = T_r)).$$

Now, since the predicate symbol ans does not appear in $\forall X(W \leftrightarrow \exists Y_1(X = T_1) \vee \ldots \vee \exists Y_r(X = T_r))$ or in $comp(P)$ we can drop the axiom $\forall X(ans(X) \leftrightarrow W)$ from $comp(P^{ans})$ arriving at

$$comp(P) \models \forall X(W \leftrightarrow \exists Y_1(X = T_1) \vee \ldots \vee \exists Y_r(X = T_r)).$$

Therefore, the thesis follows. □

By the theorem above, we have that: every time an SLDNF-resolution interpreter (e.g. a sound Prolog) terminates after providing a finite set of computed answers, we have got a *complete* set of correct answers. Actually, we have characterized *the* set of all ground correct answers. On the other hand, if $P \cup \{G\}$ has a finitely *failed* SLDNF-tree, then by Clark's theorem on soundness of negation as failure (cf. [2]) we have that G is a logical consequence of $comp(P)$ (i.e., assuming $G = \leftarrow W$, we have that $comp(P) \models\sim \exists W$, where $\exists W$ is the existential closure of W). Hence, as anticipated in the Introduction, every time an SLDNF-resolution interpreter terminates while evaluating a goal, we get a complete logical characterization of the goal.

As an easy consequence of the theorem above, we get the following corollary, which ensures that we are totally free in the choice of the non-failing finite SLDNF-tree to be used in the generation of the computed answers.

Corollary 3 *Let P be a normal program such that $comp(P)$ is consistent, and let G be a normal goal. Then, every (complete) non-failing finite SLDNF-tree for $P \cup \{G\}$ returns the same computed answers.*

Acknowledgments

I would like to thank Eugenio Omodeo for stimulating discussions and helpful suggestions.

References

[1] K. Benkerimi, J. W. Lloyd, **A Partial Evaluation Procedure for Logic Programs**, *Proc. of North American Conf. on Logic Programming*, S. K. Derbray , M. Hermenegildo, eds., MIT Press, 1990.

[2] K. L. Clark, **Negation as Failure**, *Logic and Data Bases*, H. Gallaire, J. Minker, eds, Plenum Press, New York, 1978.

[3] J. W. Lloyd, *Foundations of Logic Programming* (2nd edition), Springer-Verlag, 1987.

[4] J. W. Lloyd, J. C. Shepherdson, **Partial Evaluation in Logic Programming**, *The Journal of Logic Programming*, 11(3&4), October/November 1991.

[5] J. C. Shepherdson, **Negation as Failure II**, *The Journal of Logic Programming*, 2(3), October 1991.

[6] J. C. Shepherdson, **Negation in Logic Programming**, *Foundations of Deductive Database and Logic Programming*, J. Minker, ed., Morgan Kaufmann, 1988.