A Floyd-Hoare Method for Prolog

W. Drabent, Linköping University and Polish Academy of Sciences
e-mail: wdr@ida.liu.se
May 6, 1996

ABSTRACT: We present an inductive assertion method for proving run-time properties of logic programs. The method could be seen as a logic programming counterpart of the well-known approach of Floyd and Hoare for imperative programs. The method concerns assertions assigned to program points and makes it possible to prove that whenever the control reaches a program point the corresponding assertion is satisfied. We also show a way of augmenting the method to prove termination. An assertion (assigned to a point in a program clause) describes a set of substitutions (for the variables of the clause). Assertions may be not monotonic (i.e. not closed under substitutions).

1 Introduction

We present a method of proving run-time properties of logic programs that are executed with Prolog selection rule. By run-time properties we mean properties of LD-derivations, such properties are in general not expressible in terms of the declarative semantics. Examples of such properties are the bindings of program variables at certain program points, or the actual form of procedure calls or successes occurring during computation. The latter includes modes, avoiding occur-check, correct usage of Prolog arithmetics, variable aliasing etc.

We describe the properties by assigning assertions to program points. Program points are placed after every atom in the clauses of the program. An asserted program is correct if each assertion is satisfied whenever the control reaches the corresponding program point. An assertion describes a set of substitutions, for a correctly asserted program this set contains any substitution that may appear when the corresponding program point is reached. It is important that assertions may be not monotonic (i.e. not closed under substitutions). For instance an assertion may state that variable X is bound to a non-ground term.

In our former work [DM88, Dra88] we proposed an inductive assertion method in which assertions are assigned to procedures: to every predicate symbol there corresponds a precondition and a postcondition[1]. The method presented here usually requires more assertions but the verification conditions (the elementary proofs to be performed) are simpler. We believe that the two methods are complementary. Some properties of interest cannot be (directly) expressed in terms of pre- and postconditions for predicates. On the other hand this is a more natural way to express many properties of interest.

A stimulus to undertake this work was the paper by Colussi and Marchiori

[1] For other methods with assertions assigned to procedures, see [AM94]. They may be seen as specializations the method of [DM88]. For a example applications of that method, see [Dra94] and [RNP92].

[CM91]. They also proposed an inductive assertion method with assertions assigned to program points. However, their soundness theorem concerns only the final answers. Hence, strictly speaking, that method does not deal with run-time properties. In contrast, our soundness theorem concerns all program points of any derivations (including failing and infinite ones). The method presented here is also simpler. For further comparisons, see Section 8.

In our work we consider LD-resolution, in other words definite logic programs executed with Prolog selection rule (the leftmost atom first). Our method can be easily extended to many build-in procedures of Prolog. It is independent from the way the LD-tree is searched, so it is sound for Prolog, OR-parallelism with Prolog selection rule, Prolog with the cut etc. (It does not take into account the pruning of the LD-tree made by the cut).

Our method considers partial correctness (whenever a control point is reached then the corresponding assertion holds). In Section 7 we augment it with proving termination.

Run-time properties dealt with by our method include properties of the computed answers. In particular they include properties of the S-semantics and declarative properties. (By declarative we mean the semantics given by the notion of logical consequence or by the least Herbrand model of the program).

It should be made clear what is the application domain of methods like the one presented here. Run-time properties are in a sense of secondary importance for logic programming. A substantial part of a programmer's work can be done referring to the declarative properties only, together with reasoning about termination. Reasoning about declarative properties [Cla79, Hog81, Der93] is usually simpler than that about run-time properties. So in most cases methods like ours are useful only when one is interested in properties not expressible in terms of the declarative semantics. We may suggest a slogan that such methods are mainly for non logical properties of logic programs.

Our work may be seen as transferring to logic programming the well known inductive assertion approach of Floyd and Hoare. This transfer is far from trivial as the basic operations, respectively assignment and unification, are substantially different. From the imperative programming point of view, unification results in sophisticated forms of aliasing of variables.

Verification conditions for imperative programs concern properties of the semantic domains of variables. For example to show that $\{6|x\}\ x := x + 1\ \{2 \not| x\}$ holds one has to show that the verification condition $6|x \rightarrow 2 \not| x + 1$ is true in the domain of integers. In contrast, verification conditions in our method concern properties of unification in a domain of (possibly non ground) terms. We express such properties in the form of Hoare triples. We do not discuss methods of proving verification conditions. An attempt is presented in [CM92] where the authors introduce a proof system for the unification.

The paper is organized as follows. We begin with introducing the notion of assertions and of Hoare triples for unification. Then we introduce the proof method, show an example proof and prove the method's soundness. The following sections concern proving termination, discussion of the related work and a proposed continuation of this work.

2 Preliminaries

We use the standard notation and terminology [Llo87, Apt90]. We will use Prolog syntax in the examples. When referring to the language of programs, s, t will usually stand for terms, a, b, c for constants, f, g for function symbols, x, y, z, v for variables, p, q for predicate symbols, A, B, H for atoms. Subscripts may be used if necessary. Over-lining will be used to denote a (finite) sequence of objects. For instance $\overline{x}$ is an abbreviation for $x_1, \ldots, x_n$ for some integer $n \geq 0$, $p(\overline{t})$ abbreviates $p(t_1, \ldots, t_m)$ (where m is the arity of p), $\overline{s} = \overline{t}$ abbreviates $s_1 = t_1, \ldots, s_n = t_n$ and $\{\,\overline{x}/\overline{t}\,\}$ abbreviates $\{\,x_1/t_1, \ldots, x_n/t_n\,\}$ (for some $n \geq 0$). Sometimes we do not distinguish between a sequence and the corresponding set (and for instance write $\overline{x}$ for $\{x_1, \ldots, x_n\}$).

Vars(E) will denote the set of variables occurring in a syntactic construct (expression or substitution) E. The domain of a substitution θ will be denoted by Domain(θ). The restriction of a substitution θ to a set of variables S will be denoted $\theta|_S$. For an expression E, we will abbreviate $\theta|_{\mathrm{Vars}(E)}$ as $\theta|_E$. By a renaming we mean a substitution $\{\overline{x}/\overline{y}\}$ where $\overline{y}$ is a permutation of $\overline{x}$.

3 Assertions

Assertions are first order logical formulae that describe sets of states. It is a basic concept of most approaches to prove correctness of imperative programs. A state is a mapping of program variables into their values. In the context of logic programming such mapping is just a substitution.

To formalize the notion of assertions one has to define their language: fix its predicate and function symbols together with their fixed "standard" interpretation. The domain of the interpretation is a suitable domain of terms and substitutions (possibly augmented with natural numbers etc.). Sometimes we will refer to the language of assertions as to the metalanguage and to the language of logic programs under consideration as to the object language. For an assertion I and a substitution θ we will write $\models_\theta I$ or just $\models I\theta$ if I is true in the fixed interpretation and the valuation θ. We will also say "I holds for θ" or "$I\theta$ is true".

We will not introduce the metalanguage of assertions in a formal way. We assume that:

- The set of variables of the metalanguage is that of the object language (and is infinite).
- The function and predicate symbols of the object language are function symbols of the metalanguage. They are interpreted as themselves. Some other function symbols will be used in assertions, for example arithmetic operators $+, -$, etc. Their interpretation is as usual.
- The predicate symbols are $=$, *var*, *ground*, ... Their interpretation is as given below. New predicates may be added when needed.

  $\mathit{var}(t)\theta$ is true iff term $t\theta$ is a variable.

$ground(t)\theta$ is true iff term $t\theta$ is a ground.

$disjoint(t,s)\theta$ is true iff $t\theta$ and $s\theta$ have no common variable.

$free(\overline{x})\theta$ is true iff $\overline{x}\theta$ is a tuple of distinct variables.

$=$ denotes equality: $\models_\theta t = s$ iff $t\theta$ and $s\theta$ are the same term.

$instance(t,s)\theta$ is true iff $t\theta$ is an instance of $s\theta$.

$(t \prec s)\theta$ is true if $t\theta$ is a subterm of $s\theta$.

We require that the assertions are invariant under variable renaming (for an assertion I and a renaming η, $I\theta$ is true iff $I\theta\eta$ is). This excludes properties related to the identity of variables, like "the success instance will be `p(f(_777))`".

Note that we are able to deal with "non-monotonic" assertions (i.e. not closed under substitution).

4 Hoare triples for unification

In our method, proofs of program properties will be reduced to proving properties of unification. To express the latter we introduce a notion of a Hoare triple for unification [CM92].

Definition 4.1 A *Hoare triple for unification* is an expression:

$$\{Pre\}\ \{\!\!|\ E, F\ |\!\!\}\ \{Post\} \tag{1}$$

where E, F are expressions (of the object language) and $Pre, Post$ are assertions. Its intended meaning is: $Post$ holds after unifying E and F, provided Pre held before the unification[2]. More precisely, (1) is true iff, for every substitution θ and for any most general unifier σ of $E\theta$ and $F\theta$, if $Pre\theta$ is true then $Post\theta\sigma$ is true.

Example 4.2 The following Hoare triples for unification are true
$\{ground(t)\}\ \{\!\!|\ t, s\ |\!\!\}\ \{ground(s)\}$
$\{ground(x), ground(y')\}\ \{\!\!|\ f(x,y), f(x',y')\ |\!\!\}\ \{ground(x'), ground(y)\}$
Consider the first of them. It holds, because if term t is ground then after unifying it with s, s becomes ground. Formally: if $t\theta$ is ground for some substitution θ (describing a "current binding of variables") then for any most general unifier σ of $t\theta$ and $s\theta$ the obtained term $s\theta\sigma$ is ground.

The triple $\{\neg ground(t,s)\}\ \{\!\!|\ t, s\ |\!\!\}\ \{\neg ground(s)\}$ is in general not true. To show it, take $t = f(x,y)$ and $s = f(x',y')$ and assume that x and y' are bound to non-ground terms and x' and y to ground terms. The precondition is satisfied but s after the unification is ground. □

We are interested in two particular cases, where the expressions being unified do not have a common variable and where the second of them is an instance of the first. So we introduce some abbreviations.

[2] We will also say that unification of E and F is correct w.r.t. precondition Pre and postcondition $Post$.

Definition 4.3
We will write $\{Pre\}\ [\![t,s]\!]\ \{Post\}$ to abbreviate $\{Pre \wedge disjoint(t,s)\}\ \{\!|\, t, s\, |\!\}\ \{Post\}$ and
$\{Pre\}\ [\![t \lhd s]\!]\ \{Post\}$ to abbreviate $\{Pre \wedge disjoint(t,s) \wedge instance(s,t)\}\ \{\!|\, t, s\, |\!\}\ \{Post\}$.

We do not provide a formal method of proving Hoare triples for unification. Such a method was proposed in [CM92]. Instead we will use informal proofs, as usually in mathematics. We conclude this section with stating some properties that could be useful in such proofs. The following proposition shows that to prove $\{Pre\}\ \{\!|\, t, s\, |\!\}\ \{Post\}$ it is sufficient to consider only one mgu and that it is not necessary to consider all substitutions satisfying *Pre*. (It is sufficient to consider one element for each equivalence class of substitutions equal up to renaming).

Proposition 4.4 $\{Pre\}\ \{\!|\, t, s\, |\!\}\ \{Post\}$ is true iff for every substitution θ such that $Pre\theta$ is true, either $t\theta$ and $s\theta$ are not unifiable or $Post\theta\sigma$ is true for some mgu σ of $t\theta$ and $s\theta$.

Let η be a renaming substitution. $t\theta$ and $s\theta$ are unifiable iff $t\theta\eta$ and $s\theta\eta$ are. If $\models Pre\theta$ and σ is an mgu of $t\theta$ and $s\theta$ and $\models Post\theta\sigma$ then $\models Pre\theta\eta$ and $\models Post\theta\eta\sigma'$ for every mgu σ' of $t\theta\eta$ and $s\theta\eta$.

PROOF
If σ' is an mgu of $t\theta$ and $s\theta$ then $\sigma' = \sigma\eta$ for some renaming η. $Post\theta\sigma\eta$ holds as assertions are invariant under renaming.

Let η be a renaming. Then there exists a renaming η^{-1} such that $\eta\eta^{-1} = \varepsilon$. Let σ be an mgu of $t\theta$ and $s\theta$. We show that $\rho = \eta^{-1}\sigma$ is an mgu of $t\theta\eta$ and $s\theta\eta$. Obviously it is a unifier. Assume that $t\theta\eta\varphi = s\theta\eta\varphi$. Then $\eta\varphi$ is a unifier of $t\theta$ and $s\theta$, so there exists δ such that $\eta\varphi = \sigma\delta$. Then $\varphi = \eta^{-1}\sigma\delta = \rho\delta$; hence ρ is most general. By symmetry, if $t\theta\eta$ and $s\theta\eta$ are unifiable then $t\theta$ and $s\theta$ are. $Pre\theta\eta$ holds as the assertions are invariant w.r.t. renaming. $Post\theta\eta\rho$ holds as $\theta\eta\rho = \theta\sigma$. From the previous part of the proof it follows that $Post\theta\eta\sigma'$ holds for any mgu σ' of $t\theta\eta$ and $s\theta\eta$. □

Note that it is sufficient to consider σ that is idempotent and relevant. (If such a unifier does not exist then $t\theta$ and $s\theta$ are not unifiable). If $\{Pre\}\ \{\!|\, t, s\, |\!\}\ \{Post\}$ is $\{Pre'\}\ [\![t \lhd s]\!]\ \{Post\}$ then it is sufficient to consider σ such that $\mathrm{Domain}(\sigma) \subseteq \mathrm{Vars}(t\theta)$. (If s' is an instance of t' then there exists their mgu σ such that $t'\sigma = s'$). Let $\overline{y}$ be the variables of s. If the triple is of the form $\{Pre \wedge free(\overline{y})\}\ [\![t,s]\!]\ \{Post\}$ then it is sufficient to consider substitutions θ that do not bind $\overline{y}$ (i.e. $\overline{y} \cap \mathrm{Domain}(\theta) = \emptyset$).

5 The method

To specify run-time properties of programs we decorate their clauses with assertions:

Definition 5.1 (Asserted clause)

$$H \leftarrow \{I_0\}A_1\{I_1\}A_2 \cdots A_n\{I_n\} \qquad (n \geq 0)$$

where $I_0, \ldots, I_n$ are assertions and $H, A_1, \ldots, A_n$ are atoms, is called an *asserted* (definite) *clause*.

Expression $\{I\}\, A\, \{I'\}$, where I, I' are assertions and A is an atom, will be called an *atomic specification* (or a Hoare triple for an atom).

To specify a set of initial goals, together with their assertions we use a goal clause.

Definition 5.2 A *goal clause* is an asserted clause $\mathbf{goal} \leftarrow \{I_0\}A_1\{I_1\}A_2 \cdots A_n\{I_n\}$ where **goal** is a new symbol not occurring in clause bodies. This clause specifies a class of goals of the form $G = (A_1, \ldots A_n)\theta$ where $I_0\theta$ is true.

By an *asserted program*, we mean a finite set of asserted clauses, including goal clauses. So an asserted program corresponds to a finite definite program together with a possibly infinite set of initial goals. (We will often skip the word "asserted" whenever it does not lead to misunderstanding). We are interested in all computations of such a program, beginning with goals specified by (some) goal clause from the program. By a computation we mean any LD-derivation (successful, failing or infinite). Speaking informally, a program is (partially) correct if, whenever the control reaches an assertion, the assertion is satisfied. A formal definition is given later on below.

We propose an inductive assertion method for proving program correctness. The following definition is crucial for the method. It introduces a sufficient condition for program correctness: if the program (including its goal clauses) is *well asserted* then it is correct. (This is established as a formal result in Theorem 6.4). Checking whether a program is well asserted boils down to checking conditions (CALL) and (EXIT) for every (unifiable) pair of a body atom and a clause head of the program.

Definition 5.3 (Verification conditions: well-asserted program)

Atomic specification $\mathcal{S} = \{Pre\}A\{Post\}$ *agrees* with an asserted clause C if for some variant $C' = H \leftarrow \{I_0\} \cdots \{I_n\}$ of C $(n \geq 0)$ such that $\mathcal{S}$ and C' have no common variable the following conditions hold:

$$\{\, Pre \wedge free(\overline{y})\,\} \quad [\![\, A, H \,]\!] \quad \{\, I_0 \,\} \qquad \text{(CALL)}$$

$$\{\, Pre \wedge I_n \,\} \quad [\![\, A \triangleleft H \,]\!] \quad \{\, Post \,\} \qquad \text{(EXIT)}$$

where $\overline{y}$ are the variables occurring in C'.

A program P is *well-asserted* if every atomic specification $\mathcal{S} = \{I_{i-1}\}A_i\{I_i\}$ occurring in P agrees with every clause of P.

Obviously, conditions (CALL) and (EXIT) are trivially satisfied if A and H are not unifiable.

The intuitive explanation of (CALL) is rather obvious: I_0 should hold after unifying A (with its variables instantiated in such a way that Pre holds) with (uninstantiated) H. Condition (EXIT) is less obvious. It models a procedure return. Imagine that the procedure call $A\theta$ had been extracted from a goal and evaluated in isolation. Assume that it succeeded. (EXIT) models applying the computed answer substitution to the remaining atoms of the goal. The variables of H are bound according to I_n, the variables of the goal are bound by θ (i.e. as at the moment of the call of $A\theta$). Unifying A and H results in binding the variables of the goal as at the success of $A\theta$.

Example 5.4

We will use an abbreviation $\mathrm{Vars}(Y) \subseteq \mathrm{Vars}(Z)$ for formula $\forall V.\ (\mathit{var}(V) \land V{\prec}Y) \rightarrow V \prec Z$.

$$
\begin{array}{l}
\mathbf{goal} \leftarrow \overbrace{\{\mathit{free}(Y,Z) \land Y,Z{\not\prec}X\}}^{I_0}\, p(X,Y,Z)\, \overbrace{\{\mathrm{Vars}(Y) \subseteq \mathrm{Vars}(Z)\}}^{I_1}. \\
p([A|LA],[B|LB],[p(A,B)|LC]) \leftarrow \overbrace{\{\mathit{free}(LB,LC) \land LB,LC{\not\prec}LA\}}^{I_2} \\
\qquad\qquad p(LA,LB,LC) \\
\qquad\qquad \underbrace{\{\mathrm{Vars}(LB) \subseteq \mathrm{Vars}(LC)\}}_{I_3}. \\
p([\,],[\,],[\,]) \leftarrow \{\mathbf{true}\}
\end{array}
$$

The verification condition for this asserted program consists of the following eight Hoare triples for unification. (They are numbered with the number of the calling and the called clause). Let F stands for formula $\mathit{free}(A,LA,B,LB,LC)$.

$$\{I_0 \land F\} \left[\!\!\left[\begin{array}{l} p(X,Y,Z) \\ p([A|LA],[B|LB],[p(A,B)|LC]) \end{array} \right]\!\!\right] \{I_2\} \qquad \text{(CALL01)}$$

$$\{I_0 \land I_3\} \left[\!\!\left[\begin{array}{l} p(X,Y,Z) \ \lhd \\ p([A|LA],[B|LB],[p(A,B)|LC]) \end{array} \right]\!\!\right] \{I_1\} \qquad \text{(EXIT01)}$$

$$\{I_2 \land F'\} \left[\!\!\left[\begin{array}{l} p(LA,LB,LC) \\ p([A'|LA'],[B'|LB'],[p(A',B')|LC']) \end{array} \right]\!\!\right] \{I_2'\} \qquad \text{(CALL11)}$$

(The renamed variables are denoted with primes, the same for the assertions with their variables renamed.)

$$\{I_2 \land I_3'\} \left[\!\!\left[\begin{array}{l} p(LA,LB,LC) \lhd \\ p([A'|LA'],[B'|LB'],[p(A',B')|LC']) \end{array} \right]\!\!\right] \{I_3\} \qquad \text{(CALL11)}$$

$$\{I_0\} \left[\!\!\left[\begin{array}{l} p(X,Y,Z) \\ p([\,],[\,],[\,]) \end{array} \right]\!\!\right] \{\mathbf{true}\} \qquad \text{(CALL02)}$$

$$\{I_0 \land \mathbf{true}\} \left[\!\!\left[\begin{array}{l} p(X,Y,Z) \lhd \\ p([\,],[\,],[\,]) \end{array} \right]\!\!\right] \{I_1\} \qquad \text{(EXIT02)}$$

$$\{I_2\} \left[\!\!\left[\begin{array}{l} p(LA,LB,LC) \\ p([\,],[\,],[\,]) \end{array} \right]\!\!\right] \{\mathbf{true}\} \qquad \text{(CALL12)}$$

$$\{I_2 \land \mathbf{true}\} \left[\!\!\left[\begin{array}{l} p(LA,LB,LC) \lhd \\ p([\,],[\,],[\,]) \end{array} \right]\!\!\right] \{I_3\} \qquad \text{(EXIT12)}$$

Note that (CALL11) and (EXIT11) are just renamings of (CALL01) and (EXIT01). (EXIT02) and (EXIT12) are trivially true as the terms are ground after the unification. The triples with the postcondition **true** are tautologies. So to prove the correctness of the program it remains to prove (CALL01) and (EXIT01). Below we treat (CALL01) with some details.

Assume that the precondition $I_0 \wedge F$ holds for some θ. According to section 4 we may assume that Domain$(\theta) \subseteq \{X, Y, Z\}$. So θ binds Y, Z, A, LA, B, LB, LC to distinct variables that, except for $A\theta$, do not occur in $X\theta, [A|LA]\theta$. If σ is an relevant mgu of $X\theta$ and $[A|LA]\theta$ and if $\sigma' = \{Y/[B|LB],\ Z/[p(A\sigma, B)|LC]\}$ then $\sigma\sigma'$ is an mgu of $p(X, Y, Z)\theta$ and $p([A|LA], [B|LB], [p(A, B)|LC])\theta$.

Obviously, variables LB and LC are unbound by $\theta\sigma\sigma'$ and they do not occur in $LA\theta\sigma\sigma'$ (as $LA\theta\sigma\sigma' = LA\sigma$). Hence $\models I_2\theta\sigma\sigma'$. □

It is easy to extend our method to many built-in predicates of Prolog. This can be done by adding corresponding verification conditions to Definition 5.3. For instance, to cover var/1, nonvar/1 and ==/2 we need the following.

If $\mathcal{S}$ (an atomic specification occurring in the program) is of the form $\{I\}$ `var(t)` $\{J\}$ then J is $I \wedge \mathit{var}(t)$.

If $\mathcal{S} = \{I\}$ `nonvar`(t) $\{J\}$ then J is $I \wedge \neg \mathit{var}(t)$.

If $\mathcal{S} = \{I\}\, t{=}{=}s\, \{J\}$ then J is $I \wedge t = s$.

6 Soundness

To discuss correctness of programs we have to relate goals in LD-derivations to assertions in the programs. We introduce a notion of an asserted goal and an appropriate definition of an LD-derivation over asserted goals. This will make it possible to state precisely what does it mean that the control reaches an assertion and the assertion is satisfied.

Definition 6.1 (Asserted goal)

An *asserted goal* is a pair

$$\theta,\ \overline{\{I_0\}}A_1\overline{\{I_1\}}A_2 \cdots A_n\overline{\{I_n\}}$$

where $n \geq 0$, θ is a substitution and $\overline{\{I_i\}}$ is a sequence of assertions (for $i = 1, \ldots, n$).

We refer asserted goals to the standard notion of a goal in SLD-resolution by stating that the asserted goal above *corresponds* to the goal $(A_1, \ldots, A_n)\theta$.

Asserted goal $\theta, \{I_0\}A_1 \cdots A_n\{I_n\}$ is an *instance* of a goal clause **goal** $\leftarrow$ $\{I_0\}A_1 \cdots A_n\{I_n\}$ iff $I_0\theta$ is true. (So the goal that corresponds to an instance of a goal clause is one of those specified, in the sense of Def. 5.2, by the goal clause).

An *initial asserted goal* for a program P is an instance of a goal clause of P. (Note that the assertion sequences in G are of length 1).

Definition 6.2 (LD-resolution) Consider an asserted goal G:

$$\theta,\ \overline{\{I_0\}}A_1\overline{\{I_1\}}A_2 \cdots A_n\overline{\{I_n\}}$$

and an asserted clause C:

$$H \leftarrow \{J_0\}B_1 \cdots B_m\{J_m\}.$$

Assume that G and C have no common variable and that σ is an mgu of $A_1\theta$ and H. Then

$$\theta\sigma,\ \{J_0\}B_1 \cdots B_m\{J_m\}\overline{\{I_1\}}A_2 \cdots A_n\overline{\{I_n\}}$$

is a *LD-resolvent* of G and C.

Let P be an asserted program. An *asserted LD-derivation* is a (possibly infinite) sequence $G_0, G_1, \ldots$ of asserted goals such that G_0 is an initial asserted goal for P and G_i is a resolvent of of G_{i-1} and a standardized apart asserted clause C_i of the program, for $i = 1, 2, \ldots$. (A technical detail: we do not require that a derivation is a maximal such sequence).

The correspondence with the standard definition of SLD- (LD-) resolution is obvious. If $G_0, G_1, \ldots$ is an asserted LD-derivation and G_i corresponds to a ("not asserted") goal G'_i, for $i = 0, 1, \ldots$, then $G'_0, G'_1, \ldots$ is an LD-derivation. Conversely, for a given program, let $G'_0, G'_1, \ldots$ be an LD-derivation and let G_0 be an initial asserted goal that corresponds to G'_0. Then there exists an asserted LD-derivation $G_0, G_1, \ldots$ whose goals correspond, respectively, to $G'_0, G'_1, \ldots$.

Now we are ready to formalize the concept of the control reaching an assertion. As we mainly deal with asserted clauses, goals and derivations, we will usually skip the word "asserted". Note that if $\theta, \overline{\{I_0\}}A_1 \cdots A_n\overline{\{I_n\}}$ is a goal in an LD-derivation where $\overline{\{I_l\}}$ is $\{I_{l,1}\} \cdots \{I_{l,m_l}\}$ (for $0 \leq l \leq n$) then each of assertions $\{I_{l,1}\} \cdots \{I_{l,m_l-1}\}$ is the last assertion of some clause of the program.

Consider a goal $G = \theta,\ \{I_{0,1}\} \cdots \{I_{0,l}\}A_1 \cdots A_n\overline{\{I_n\}}$ in an LD-derivation (thus $l > 0,\ n \geq 0$). Note that it can be represented as $G = \theta, G_1 \cdots G_l \cdots G_m$ where each G_i $(i = 1, \ldots, m)$ is a suffix of a (renamed) program clause, $G_j = \{I_{0,j}\}$ for $j = 1, \ldots, l-1$ and $G_l = \{I_{0,l}\}\, A_1 \cdots A_k\, \{I\}$ (for some I and $k \leq n$). As the selected goal of G is A_1, we may say that in G the control reaches $\{I_{0,l}\}$. As for $j = 1, \ldots, l-1$ $\{I_{0,j}\}$ is the last assertion of a clause, G may be understood as corresponding to a success of this clause. So it is natural to say that in goal G the control reaches assertions $\{I_{0,1}\}, \ldots, \{I_{0,l}\}$.

Definition 6.3 (Correct program)
An asserted program P is correct iff for any goal $\theta,\ \overline{\{I_0\}}A_0\overline{\{I_1\}} \cdots A_m\overline{\{I_m\}}$ $(m \geq 0)$ of any its derivation, $\overline{\{I_0\}}\theta$ holds (i.e. for each assertion I in $\overline{\{I_0\}}$, $\models I\theta$).

Theorem 6.4 (Soundness)
Every well-asserted program is correct.

PROOF
Let $G_0, G_1, \ldots$ be the derivation. We may assume that the mgu's used in the derivation are relevant. The proof is by induction. The base case of G_0 is obvious. Assume that the theorem holds for $G_0, \ldots, G_{k-1}$.

Let G_{k-1} be $\theta, \overline{\{I\}}\{Pre\}A\{Post\} \cdots$ and G_k be a resolvent of G_{k-1} and a clause $C = H \leftarrow \{I_0\} \cdots \{I_n\}$ $(n \geq 0)$. Hence $G_k = \theta\sigma, \{I_0\} \cdots \{I_n\}\{Post\} \cdots$. Obviously specification $\{Pre\}A\{Post\}$ occurs in the program. From the respective (CALL) condition we obtain that $I_0\theta\sigma$ holds (as $Pre\theta$ holds, the variables of C are distinct from those of G_{k-1} and σ is an mgu of A and H). This completes the inductive step when $n > 0$; in such a case G_k begins with a single assertion $\{I_0\}$.

If $n = 0$ then $G_k = \rho, \{J_1\} \cdots \{J_l\} \cdots$ where $\rho = \theta\sigma$, J_1 is I_0 and J_2 is $Post$. We show by induction that $J_2\rho, \ldots, J_l\rho$ hold. Let $1 \leq i < l$, assume that $J_i\rho$ holds. We show that $J_{i+1}\rho$ holds. As discussed previously, J_i is the last assertion of some clause of P.

There exist two consecutive goals

$$\begin{array}{rcl} G & = & \rho_0, \ \overline{\{I\}}\{Pre_B\}B\{Post_B\}\overline{L} \\ G' & = & \rho_1, \ \cdots\{J_i\}\{Post_B\}\overline{L} \end{array}$$

in the derivation $G_0, \ldots, G_{k-1}$ such that $\overline{L}$ is a sequence of assertions and atoms, G' is a resolvent of G and $C = H' \leftarrow \cdots \{J_i\}$ and

$$G_k \ = \ \rho, \ \{J_1\} \cdots \{J_i\}\{Post_B\}\overline{L}$$

(hence $Post_B$ is J_{i+1}). Let us consider the last G, G' satisfying these conditions.

We are going to construct a substitution α that, in a sense, combines the bindings in G of the variables of $\{Pre_B\}B\{Post_B\}$ and the bindings in G_k of the variables of C.

Let $\overline{x}$ be the variables occurring in G and $\overline{y}$ the variables occurring in C. Obviously $\overline{x} \cap \overline{y} = \emptyset$. Substitution ρ can be represented as a union of two disjoint substitutions $\rho = \rho' \cup \rho''$ where $\rho' = \rho|_{\overline{x}}$. Notice that $\rho''|_{\overline{y}} = \rho|_{\overline{y}}$.

Let η be a renaming substitution for the variables of $\overline{y}\rho''$ into the variables not occurring in the derivation.

$J_i\rho$ holds, hence $J_i\rho''\eta$ holds (as assertions are invariant w.r.t. variable renaming). $Pre_B\rho_0$ holds from the inductive assumption. $disjoint(\overline{x}\rho_0, \overline{y}\rho''\eta)$ holds due to the choice of η. So for substitution $\alpha = \rho_0 \cup \rho''\eta$ the precondition in the condition (EXIT) for $\{Pre_B\}B\{Post_B\}$ and C holds.

Now we construct an mgu for $B\alpha$ and $H'\alpha$ (this means for $B\rho_0$ and $H'\rho''\eta$). We have $\rho = \rho_0\delta$ where δ is the composition of mgu's used in the derivation between G and G_k. Obviously, δ is a unifier of $B\rho_0$ and H', we also have $H'\delta = H'\rho = H'\rho''$. Hence $B\rho_0\delta = H'\rho''$ and $B\rho_0\delta\eta = H'\rho''\eta$. Thus $H'\alpha = H'\rho''\eta$ is an instance of $B\alpha = B\rho_0$ and $\beta = (\delta\eta)|_{B\alpha}$ is the required mgu.

From (EXIT) it follows that $Post_B\alpha\beta$ holds. Due to standardizing apart, we have $\mathrm{Vars}(\delta\eta) \cap \overline{x} \subseteq \mathrm{Vars}(B\rho_0)$. Hence $\beta = (\delta\eta)|_{\overline{x}}$. Notice that $Post_B\alpha\beta$ is $Post_B\rho_0\delta\eta$ which is $Post_B\rho\eta$. Remember that $Post_B$ is J_{i+1}. So from the assumption that $J_i\rho$ holds we showed that $J_{i+1}\rho\eta$ holds. Thus $J_{i+1}\rho$ holds, this completes the proof. □

7 Termination

To prove termination one usually employs a kind of measure of atoms, often called a level mapping. We present a method similar to that introduced by Apt and

Pedreschi [AP93]. It uses a level mapping $|\ |$ that is is a function from ground atoms to natural numbers. We extend $|\ |$ to non-ground atoms by stating

$$|A| = \sup\{\, |A\theta| : A\theta \text{ is ground}\,\}$$

for an arbitrary atom A. So $|A|$ is a natural number or $|A| = \omega$.

For a fixed level mapping $|\ |$ we will add $|\ |$ as a function symbol to our meta-language of assertions (with the obvious interpretation). We transfer the concept of an acceptable program [AP93] to our framework.

Definition 7.1 (Acceptable program)

An asserted program P is *acceptable* with respect to a level mapping $|\ |$ iff for every its clause $H \leftarrow \{J_0\}B_1 \cdots B_m\{J_m\}$, for $i = 1, \ldots, m$,

$$J_{i-1} \rightarrow |H| > |B_i|$$

holds (for any substitution).

The definition may be informally explained as follows. We are interested in the measure of the atoms selected in the LD-derivation. Assertion J_{i-1} describes the bindings of B_i when it is selected. So we are not interested in bindings not satisfying J_{i-1}.

Theorem 7.2 (Termination 1)

All the LD-derivations of an acceptable correct program are finite.

In the proof we will use multisets over $\omega + 1$ (the set of natural numbers with added ω). Such multisets will be values of our level mapping for goals. We will use a multiset ordering $<$ obtained in a standard way from the usual ordering of $\omega + 1$. For definitions, see [AP93] or [Kun89].

Definition 7.3 (Multiset level mapping for goals)

$$|\,\overline{\{I\}}A\,| = \sup\{\, |A\sigma| : \models \overline{\{I\}}\sigma\,\}$$

(where by $\models \overline{\{I\}}\sigma$ we mean that $\models J\sigma$ for any element $\{J\}$ of sequence $\overline{\{I\}}$).

$$||\,\theta,\ \overline{\{I_0\}}A_1\overline{\{I_1\}}A_2 \cdots A_n\overline{\{I_n\}}\,|| = \mathrm{bag}(\,|\overline{\{I_0\}}A_1\theta|, \ldots, |\overline{\{I_{n-1}\}}A_n\theta|\,)$$

where $\mathrm{bag}(x_1, \ldots, x_n)$ is the multiset with elements $x_1, \ldots, x_n$.

PROOF of the Theorem

Consider an acceptable correct program P. Let

$$G =\ \theta,\ \overline{\{I_0\}}A_1\overline{\{I_1\}}A_2 \cdots A_n\overline{\{I_n\}}$$

be a goal in a derivation of P. As the program is correct, $\models \overline{\{I_0\}}\theta$. Let

$$G' = \theta\sigma,\ \{J_0\}B_1 \cdots B_m\{J_m\}\overline{\{I_1\}}A_2 \cdots A_n\overline{\{I_n\}}$$

be its successor in the derivation, obtained by resolving G with a clause $C = H \leftarrow \{J_0\}B_1 \cdots B_m\{J_m\}$ using an mgu σ. We show that $||G|| > ||G'||$.

Let $1 \leq i \leq m$. Consider a substitution φ such that $\models J_{i-1}\theta\sigma\varphi$. Then $|B_i\theta\sigma\varphi| < |H\theta\sigma\varphi| = |A_1\theta\sigma\varphi| \leq |A_1\theta| \leq |\overline{\{I_0\}A_1}|$. (The first inequality holds as the program is acceptable, the others follow from the definition of $|\ |$ for non-ground atoms and for $\overline{\{I_0\}A_1}$ respectively). Hence $|\,\{J_{i-1}\}\,B_i\theta\sigma| < |\overline{\{I_0\}A_1}|$.

Note that for $j = 2, \ldots, n$, $|\overline{\{I_{j-1}\}A_j}\theta\sigma| \leq |\overline{\{I_{j-1}\}A_j}\theta|$. Hence $||G'|| < ||G||$, by the definition of the multiset ordering. The derivation is finite, as in the multiset ordering there do not exist infinite decreasing sequences. □

8 Related work

A stimulus to undertake this work was the paper by Colussi and Marchiori [CM91]. They also proposed an inductive assertion method with assertions assigned to program points. However, their soundness theorem concerns only the final answers of successful derivations. Hence, strictly speaking, the method does not deal with run-time properties. In contrast, our soundness theorem concerns all program points of any derivations (including failing and infinite ones).

The method of [CM91] requires that, in a sense, an assertion should be able to describe the whole state of computation, not only the binding of those variables that occur in the clause. This boils down to adding extra arguments to the predicates and extra variables to the clauses. (It is unclear to which variables in an LD-derivation these new variables should be referred. Hence the author believes that for the framework of [CM91] it is impossible to define what does it mean that an assertion at a program point holds.)

A restricted version of the method of [CM91] is presented in [AM94] (conf. well-asserted programs in Section 6 of [AM94]). That method does not use additional variables and arguments and does prove run time properties. The verification conditions of our method are however simpler than the verification conditions of [AM94, Def. 6.7]. Also our notion of asserted query, used to formally define the correctness of the program, is simpler. We conjecture that our method is not less general.

9 Future work

In the presented method we have to prove a verification condition for each (unifiable) pair of a body atom and a clause head. The number of conditions to prove can be reduced by using additional assertions, namely a pre- and a post-condition for each predicate symbol. This would result in a verification condition per each body atom and each clause head of the program. (Hence $b + h$ conditions instead of $b * h$).

Such a proof method could be seen as a simplification of that of [DM88] (where the only assertions are pre- and post-conditions for predicates). Instead of a rather sophisticated valuation sequences of [DM88], the proposed approach would use additional assertions for program points. This would result in simpler proofs.

Specifying run-time properties by assertions (in a way presented here) has a drawback: some properties cannot be expressed. For instance, we cannot express a fact that executing $p(t)$ does not instantiate any variables of t; due to this our method fails for the problem from Example 6.6 in [AM94]. To be able to deal with such properties one may employ binary assertions, as done in [DM88]. (The opinion stated in [AM94] that this example cannot be dealt with by the method of [DM88], is not true.) Binary assertions refer to a pair of states, instead of a single state. For example, we can use an assertion stating that the success instance of $p(t)$ is a variant of its call instance. Generalization of the presented method to binary assertions is a subject of further work.

ACKNOWLEDGMENTS:

This research was partly supported by Swedish Research Council for Engineering Sciences (dnr 221-93-942) and by Polish Academy of Sciences.

References

[AM94] K.R. Apt and E. Marchiori. Reasoning about Prolog programs: from modes through types to assertions. *Formal Aspects of Computing*, 6(6A):743–764, 1994.

[AP93] K. Apt and D. Pedreschi. Reasoning about termination of pure Prolog programs. *Information and Computation*, 106(1):109–157, 1993.

[Apt90] K. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, chapter 10, pages 493–574. Elsevier Science Publishers B.V., 1990.

[Cla79] K. L. Clark. Predicate logic as computational formalism. Technical Report 79/59, Imperial College, London, December 1979.

[CM91] L. Colussi and E. Marchiori. Proving correctness of logic programs using axiomatic semantics. In K. Furukawa, editor, *8th International Conference on Logic Programming*, pages 629–642. MIT Press, 1991.

[CM92] L. Colussi and E. Marchiori. A predicate transformer for unification. In Krzysztof Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 67–81. The MIT Press, 1992.

[Der93] P. Deransart. Proof methods of declarative properties of definite programs. *Theoretical Computer Science*, 118:99–166, 1993.

[DM88] W. Drabent and J. Małuszyński. Inductive assertion method for logic programs. *Theoretical Computer Science*, 59:133–155, June 1988. Special issue with selected papers from TAPSOFT'87, Pisa.

[Dra88] W. Drabent. On completeness of the inductive assertion method for logic programs. Unpublished note, Institute of Computer Science, Polish Academy of Sciences, May 1988.

[Dra94] W. Drabent. A proof using the inductive assertion method. Unpublished note, available as `http://www.ida.liu.se/~wlodr/nice.ps`, November 1994.

[Hog81] C. J. Hogger. Derivation of logic programs. *Journal of ACM*, 28(2):372–392, 1981.

[Kun89] K. Kunen. Signed data dependencies in logic programs. *J. Logic Programming*, 7(3):231–245, November 1989.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, second, extended edition, 1987.

[RNP92] Y. Rouzaud and L. Nguyen-Phuong. Integrating modes and subtypes into a Prolog type-checker. In K. Apt, editor, *Logic Programming. Proc. of the Joint Int. Conference and Symposium*, pages 85–97. MIT Press, 1992.