Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

Proving Run-Time Properties of General Programs w.r.t Constructive Negation

E. Marchiori

Computer Science/Department of Software Technology

CS-R9245 1992

Proving Run-Time Properties of General Programs w.r.t. Constructive Negation

Elena Marchiori
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract

In these notes we study run-time properties of general programs w.r.t. constructive negation, i.e. termination and properties of the form of the arguments of the literals selected during the execution. We consider here SLD-CNF resolution, i.e. resolution with constructive negation and arbitrary selection rule, and LD-CNF resolution, i.e. resolution with constructive negation and Prolog selection rule. We show that the class of programs which terminate for all ground goals for arbitrary (resp. Prolog) selection rule coincides with the so-called *acyclic* (resp. *acceptable*) programs, and that SLD-CNF (resp. LD-CNF) resolution is sound and complete w.r.t. Clark's semantics for bounded goals and acyclic (resp. acceptable) programs. These results are applied to the study of run-time properties of general programs: two proof methods are introduced and their soundness is proven respectively w.r.t. SLD-CNF and LD-CNF resolution.

1991 Mathematics Subject Classification: 68Q40, 68Q60, 68T15.
1991 CR Categories: F.3.1., F.4.1, I.2.3.
Keywords and Phrases: Prolog programs, constructive negation, nonmonotonic reasoning, termination, strongest postcondition, partial correctness.
Note This research was partly supported by Esprit BRA 6810 (Compulog 2).

1 Introduction

Motivation

In this paper we investigate run-time properties of general programs w.r.t. constructive negation. Run-time properties are related to the execution of the program: such properties include termination or the form of the arguments of the literals selected during the execution. SLD-CNF resolution has been introduced by Chan [Chan88] and it allows to handle general non-ground negative goals. We distinguish between SLD-CNF resolution, i.e. resolution with constructive negation and arbitrary selection rule, and LD-CNF resolution, i.e. resolution with constructive negation and Prolog selection rule.

When SLD-CNF resolution is assumed, we consider *acyclic programs*, a natural class of locally stratified programs studied in [Cav89] and [ApBe91]. When LD-CNF resolution is assumed, we consider *acceptable programs*, a class of programs studied in [ApPe90] and [ApPe91]. Acyclic (resp. acceptable) programs enjoy the nice property of terminating for arbitrary (resp. Prolog) selection rule for a large class of bounded goals that includes all ground goals. The converse holds only under a *very* restricted assumption that no non-ground negative literal can be selected during the execution of a goal. In this paper we show how these results can be improved by using constructive negation instead of negation as failure. By considering SLD-CNF (resp. LD-CNF) resolution, a characterization of acyclic (resp. acceptable) programs in terms of computing can be made, and completeness for all bounded goals can be achieved. It has been shown in [ApBe91] how various forms of temporal reasoning can be naturally described by means of acyclic programs, as exemplified by the acyclic program YSP that formalizes the so called Yale Shooting Problem ([HaMcDe87]). We show that with constructive negation more goals can be successfully computed for the program YSP than by means of SLD-NF resolution.

These results are applied to the study of partial correctness of general programs w.r.t. constructive negation. Partial correctness of positive logic programs has been studied by various authors. There have been considered both *declarative properties*, i.e. properties of a suitable model of the program ([BoCo89], [Der89], [AnGa91]) and *run-time properties*, i.e. properties regarding the actual form of the arguments of a goal during its execution ([DrMa87], [CoMa91], [CoMa92]). The only approach we know to prove partial correctness of general programs deals with declarative properties and is the one introduced by Ferrand and Deransart in [FeDe92]: assertions are associated to every predicate defined in the program and the method ensures that they are implied by a suitable model of the program. In contrast, we are not aware of any work on run-time properties of general programs w.r.t. constructive negation. Run-time properties of programs containing negative literals are not easy to prove. In fact a resolvent of a negative subgoal $\leftarrow \neg A$ refers to the (finite) tree of $\leftarrow A$. Thus to prove a property of $\neg A$ one has to refer to a property of A and show that A terminates.

In this paper two proof methods are proposed to prove run-time properties of those general programs P such that their negative part P^- forms an acyclic (resp. acceptable) program. We consider properties that can be expressed by means of monotonic assertions, for instance *ground* or $\neg var$. We adopt a formalization in terms of Hoare-like triples, $\{pre(L)\}L\{post(L)\}$, called specifications, whose intended meaning is that if the assertion $pre(L)$ holds before the execution of the goal $\leftarrow L$, then the assertion $post(L)$ holds at the end of every successful computation. The soundness of the methods w.r.t. SLD-CNF (resp. LD-CNF) resolution is proven, using the property that SLD-CNF (resp. LD-CNF) resolution is sound and complete for bounded goals and acyclic (resp. acceptable) programs.

Preliminaries

We recall some notions and definitions on SLD-CNF resolution, taken mainly from [Chan88] and [Prz89].

Query answering in logic programming can be viewed as a translation procedure that transforms a given formula Q (the query) into an equivalent equality formula F_Q (an answer) w.r.t. the considered semantics. An *equality formula* is a formula that does not contain any predicate symbols other than equality $=$, $TRUE$ and $FALSE$.

If a refutation of a goal $G = \leftarrow Q$ in the program P does not involve the selection of any negative literals and $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ is a c.a.s. for $P \cup \{G\}$ then

$$comp(P) \models Q \leftarrow \mathcal{A},$$

where $comp(P)$ denotes the completion of the program P and $\mathcal{A}$ denotes the equality formula $(x_1 = t_1 \wedge \ldots \wedge x_n = t_n)$ relative to the substitution θ. We call $\mathcal{A}$ an answer to Q. Let $\mathcal{A}_1$, $\ldots, \mathcal{A}_k$, $k \geq 0$, be all answers to the query Q, then

$$comp(P) \models Q \equiv \mathcal{A}_1 \vee \ldots \vee \mathcal{A}_k.$$

If there are no answers, i.e. $k = 0$, then $comp(P) \models Q \equiv FALSE$ allows to deduce $\neg Q$. This is negation as failure. If an answer is equivalent to $TRUE$ then $comp(P) \models Q \equiv TRUE$ allows to deduce that $\neg Q$ is false. When the disjunction of answers is not equivalent to $TRUE$ then we cannot deduce that $\neg Q$ is false. This is why the usual implementation of negation is unsound. In constructive negation the rule

$$comp(P) \models \neg Q \equiv \neg(\mathcal{A}_1 \vee \ldots \vee \mathcal{A}_k)$$

is used. It allows to return the negation of answers to Q as answers to $\neg Q$. The following notions are useful.

An *extended literal* L is a literal of one of the following forms:

1) $p(s_1, \ldots, s_n)$, $\neg p(s_1, \ldots, s_n)$,

2) $s = t$, $\forall(s \neq t)$,

where $s, t, s_1, \ldots, s_n$ are terms, p is not an equality relation and $\forall$ quantifies some (perhaps none) of the variables occurring in the inequality. We call extended literals in 1) *simple literals* and extended literals in 2) *constraints* or (respectively positive or negative) *equality literals.*

A *primitive inequality* is a negative equality literal which is satisfiable but not valid.

A *general program* is a finite set of universally quantified clauses of the form $A \leftarrow L_1, \ldots, L_m$, where $m \geq 0$, A is a positive simple literal and L_i's are extended literals.

A formula S is called a *simple equality* if it is of the form $\exists(L_1 \wedge \ldots \wedge L_n)$, $n \geq 0$, where the L_i's are equality literals and $\exists$ quantifies over some (perhaps none) of the variables occurring in the L_i's. An empty conjunction is identified with true.

The *full answer substitution* to Q (w.r.t. a program P), denoted by F_Q, is the disjunction of the answers for $P \cup \{\leftarrow Q\}$, which are simple equality formulas generated at all success nodes of the resolution tree.

We refer to Chan [Chan88] for the definition of *normalized* answers and negation of an answer. Thus we assume that answers are normalized and their negation is computed by a suitable procedure, namely that described by Chan in [Chan88].

We call a goal G *reduced* if it is empty or it contains only primitive inequalities.

The definition of SLD-CNF tree we consider is a modification of the definition of SLDNF tree by Apt and Doets in [ApDo92] in which the constructive negation is used. To resolve negative literals, subsidiary trees are constructed, but their construction is no longer viewed as an atomic step. The trees are defined in a top-down manner by constructing their branches in parallel. An *SLD-CNF tree* is defined as the limit of a sequence $\tau_0, \ldots, \tau_n, \ldots$, such that

- every τ_i is a *pre SLD-CNF tree*, i.e. a set $\mathcal{T}$ of trees whose nodes are goals together with (if not empty) one selected literal; $\mathcal{T}$ contains one *main* tree and a function *subs* assigning to some nodes of the trees with selected negative simple literal $\neg A$ a (subsidiary) tree in $\mathcal{T}$ with root $\leftarrow A$.

- τ_0 is an *initial pre SLD-CNF tree*, i.e. it contains only one tree, with a single node which is the considered goal together with a selected literal.

- τ_{i+1} is an extension of τ_i.

The definition of extension of a pre SLD-CNF tree $\mathcal{T}$ is given as follows. We refer to [Chan88] for the description of the procedure to *normalize* answers and to negate them. Let $G - \{L\}$ denote the goal obtained removing L from G.

Mark the reduced goals as successful; if $G = L_1, \ldots, L_k$ is reduced then its associated answer (to be normalized) is the simple equality formula defined as $\exists(x_1 = x_1\theta \wedge \ldots \wedge x_n = x_n\theta \wedge L_1 \wedge \ldots \wedge L_k)$, where $x_1, \ldots, x_n$ are the variables of G, θ is the composition of the previously applied m.g.u.'s and $\exists$ quantifies over all free variables not in G.

For every unmarked leaf G with selected literal L, in a tree $T \in \mathcal{T}$ do

- If L is a positive simple literal, say A, then

  if G has no resolvents then mark G as *failed*, otherwise add all the resolvents as the sons of G in T and select literals in non-empty resolvents.

- If L is a negative simple literal, say $\neg A$, then

  if $subs(G)$ is undefined then the tree T' with the single node $\leftarrow A$ is added to $\mathcal{T}$ and $subs(G)$ is set to T';

  if $subs(G)$ is defined then the immediate descendents of G in T are the derived resolvents obtained as follows:

  if $subs(G)$ has no answers (is finitely failed) then $G - \{L\}$ is the single immediate descendant of G in T, with one selected literal;

  if $subs(G)$ is a successful tree then if the disjunction of its answers is equivalent to $TRUE$ then the goal G is marked as *failed*. Otherwise let $A_1, \ldots, A_n$ $(n > 0)$ be its normalized answers and let $NA_1 \vee \ldots \vee NA_p$ the disjunction of simple equalities obtained by negating the disjunction $(A_1 \vee \ldots \vee A_n)$: then for every $j \in [1, p]$, the goal obtained by G replacing L with the NA_j, with one selected literal, is a son of G in T;

- if L is a constraint then

  if it is an equality $r = s$ then if r and s have no unifier then G is marked as *failed*, otherwise the immediate descendant of G in T is $(G - \{L\})\theta$, with one selected literal, where $\theta = mgu(r, s)$;

  if it is an inequality $\forall(r \neq s)$ then if it is valid then the immediate descendant of G in T is $G - \{L\}$ with one selected literal; if the inequality is unsatisfiable then G is marked as *failed*. Primitive inequalities cannot be selected.

The definition of LD-CNF tree is analogous to the previous one, but a fixed selection rule is considered, namely that corresponding to the selection of the leftmost possible literal, where a literal is called *possible* if it is not a primitive inequality. We call this selection rule *Prolog selection rule.*

We consider acyclic programs ([Cav89], [ApBe91]) and acceptable programs ([ApPe90], [ApPe91]). We recall some relevant notions and results, obviously generalized to programs containing contraints.

A general program P is called *terminating* if all SLD-CNF derivations of P starting in a ground goal are finite.

A general program P is called *left terminating* if all LD-CNF derivations of P starting in a ground goal are finite.

We say that a literal is *ground* or *variable-free* if it does not contain any variable.

A *level mapping* is a function $|\ |$ from ground literals to natural numbers s.t. $|\neg A| = |A|$. We assign level mapping zero to a constraint. It is convenient to assume that for simple literals the level mapping is strictly greater than zero.

A general program P is called *acyclic w.r.t. a level mapping* $|\ |$ if for all ground instances $H \leftarrow B_1, \ldots, B_m$ of clauses of P we have that $|H| > |B_i|$ holds for all $i \in [1, m]$. A general program P is called *acyclic* if there exists a level mapping $|\ |$ s.t. P is acyclic w.r.t. $|\ |$.

A literal L is called *bounded* w.r.t. a level mapping $|\ |$ if $|\ |$ is bounded on the set $[L]$ of variable-free instances of L. If L is bounded then $|[L]|$ denotes the maximum that $|\ |$ takes on $[L]$. Then we say that L is bounded by l if $l \geq |[L]|$. A general goal $G = \leftarrow L_1, \ldots, L_n$ is called bounded w.r.t. $|\ |$ if every L_i is bounded w.r.t. $|\ |$, for all $i \in [1, n]$. If G is bounded then $|[G]|$ denotes the (finite) multiset (see [Der87]) consisting of the natural numbers $|[L_1]|, \ldots, |[L_n]|$.

The following results hold.

Lemma 1.1 ([ApBe91]) *Let $|\ |$ be a level mapping and L a bounded literal. Then, for every substitution θ, $L\theta$ is bounded and $|[L\theta]| \leq |[L]|$.*

Lemma 1.2 ([ApBe91]) *Let P be acyclic w.r.t. $|\ |$. Then, for every clause $H \leftarrow L_1, \ldots, L_n$ of P and for every substitution θ we have: if $H\theta$ is bounded then $L_i\theta$ is bounded and $|[L_i\theta]| < |[H\theta]|$ for all $i \in [1, n]$.*

Lemma 1.3 ([Bezem90]) *Let G be a goal, C a clause and θ a substitution. If $G\theta$ and C have a SLD resolvent G', then G and C have a SLD resolvent G'' s.t. $G' = G''\theta'$ for some substitution θ'.*

The previous lemmas continue to hold in presence of constraints, as in SLD-CNF resolution the execution of a selected constraint always terminates and its level mapping is by definition 0.

In the following definitions taken from [ApPe91] we assume the notion of general program previously given and LD-CNF resolution.

Let p and q be relations. We say that p *refers* to q iff there is a clause in P that uses p in its head and q in its body. We say that p *depends on* q iff (p, q) is in the reflexive, transitive closure of the relation *refers to*.

Neg_P denotes the set of relations in P which occur in a negative literal in a body of a clause from P and Neg_P^* denotes the set of relations in P on which the relations in Neg_P depend on. P^- denotes the set of clauses in P in whose head a relation from Neg_P^* occurs.

Let $|\ |$ be a level mapping for P and I a model of P whose restriction to the relations from Neg_P^* is a model of $comp(P^-)$. P is called *acceptable w.r.t.* $|\ |$ *and* I if for all ground instances $H \leftarrow L_1, \ldots, L_n$ of clauses of P we have that $|H| > |B_i|$ holds for all $i \in [1, \overline{n}]$, where $\overline{n} = min(\{n\} \cup \{i \in [1, n] \mid I \not\models L_i\})$. P is called *acceptable* if it is acceptable w.r.t. some level mapping and a model of P whose restriction to the relations from Neg_P^* is a model of $comp(P^-)$.

Let $G = \leftarrow L_1, \ldots, L_n$ be a ground general goal, $|\;|$ a level mapping and I a model of P whose restriction to the relations from Neg_P^* is a model of $comp(P^-)$. Then we associate to G the multiset $|G|_I = bag(|L_1|, \ldots, |L_{\overline{n}}|)$, where $\overline{n} = min(\{n\} \cup \{i \in [1, n] \mid I \not\models L_i\})$. We associate to a general goal G a set of multisets $|[G]|_I = \{|G'|_I \mid G'$ is a ground instance of $G\}$. G is called *bounded* by k w.r.t. $|\;|$ and I if $k \geq l$ for $l \in \cup|[G]|_I$, where $\cup|[G]|_I$ denotes the set-theoretic union of the elements of $|[G]|_I$. G is called *bounded* w.r.t. $|\;|$ and I if for some k it is *bounded* by k w.r.t. $|\;|$ and I.

It is immediate to check that Lemma 1.1 holds also when $\leftarrow L$ is a goal bounded w.r.t. a level mapping $|\;|$ and a model I.

2 Procedural Semantics of Acyclic Programs

We study here procedural semantics of acyclic programs w.r.t. SLD-CNF resolution. We show that acyclic programs characterize the class of programs that terminate for all ground goals, for arbitrary selection rule.

Theorem 2.1 *Let P be an acyclic program and G a bounded goal. Then every SLD-CNF tree for $P \cup \{G\}$ contains only bounded goals and is finite.*

Proof. The proof is similar to that of Theorem 4.1 of [ApBe91]. Let G be a bounded goal and L its selected literal. We distinguish the following three cases.

Case 1: L is positive. By Lemmas 1.1 and 1.2, it follows that the resolvent G' of G is bounded and that $|[G']|$ is smaller than $|[G]|$ in the multiset ordering.

Case 2: L is negative, say $\neg A$. Then $subs(G)$ has root $\leftarrow A$ which is obviously bounded and $|[\leftarrow A]|$ is smaller or equal than $|[G]|$ in the multiset ordering (since $|A| = |\neg A|$). Moreover, every resolvent G' of G (if any) is bounded and $|[G']|$ is smaller than $|[G]|$ in the multiset ordering, since it is obtained from G by replacing the selected literal with a (possibly empty) conjunction of constraints, whose level mapping is by assumption equal to zero.

Case 3: L is a constraint. Then the resolvent G' of G is obtained by removing the selected literal and applying the relative (if any) substitution. Thus G' is a bounded goal and by Lemma 1.1 $|[G']|$ is smaller than $|[G]|$ in the multiset ordering.

Now note that there can be only finitely many consecutive selections of negative literals and use the fact that the multiset ordering is well founded. □

In [Chan88] it is stated that SLD-CNF resolution is sound and complete with respect to Clark's semantics for finite trees. So by virtue of Theorem 2.1 the following result holds.

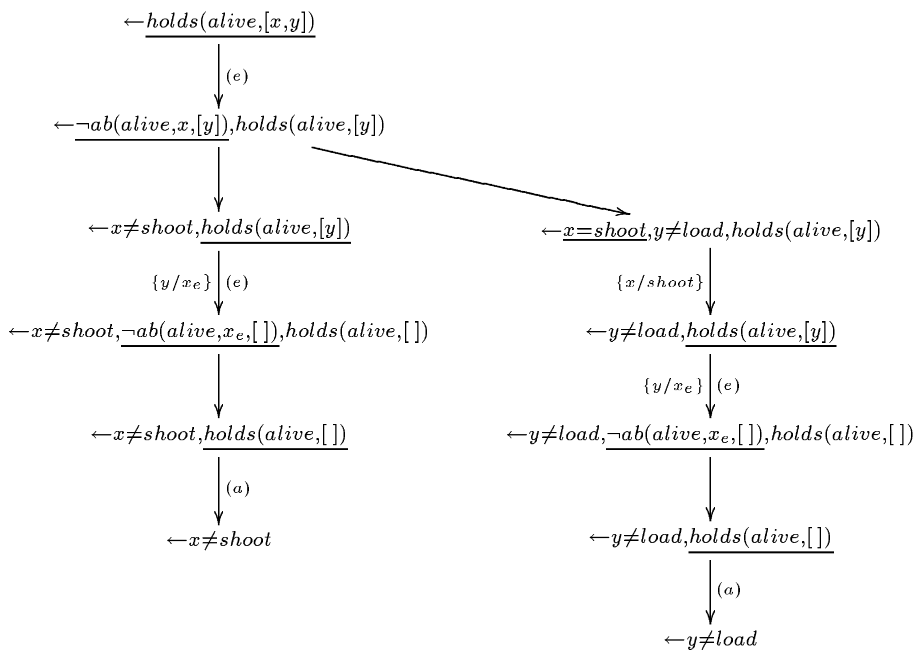
Corollary 2.2 *SLD-CNF resolution is sound and complete w.r.t. Clark's completion for bounded goals and acyclic programs.*

Example 2.3 Consider the acyclic program YSP presented by Apt and Bezem in [page339][ApBe91], which formalizes the Yale Shooting Problem.

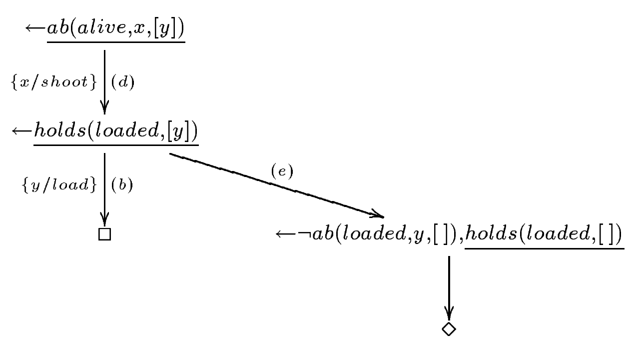
(a) `holds(alive, [])` $\leftarrow$
(b) `holds(loaded,[load|Xs])` $\leftarrow$
(c) `holds(dead,[shoot|Xs])` $\leftarrow$

```
   holds(loaded,Xs)
(d)  ab(alive,shoot,Xs) ←
   holds(loaded,Xs)
(e)  holds(Xf,[Xe|Xs]) ←
   ¬ ab(Xf,Xe,Xs),
   holds(Xf,Xs)
```

In this article it has been proved that the formula $holds(alive,[x,y])$ is equality definable as $\neg(x = shoot \wedge y = load)$ and that the goal $\leftarrow holds(alive,[x,y])$ is bounded. Then by Corollary 2.2 $\leftarrow holds(alive,[x,y])$ has a SLD-CNF tree with full answer substitution equivalent to $\neg(x = shoot \wedge y = load)$. The following is one such tree, where the selected literal is underlined, and where $\Box$ denotes success and $\Diamond$ failure.



where $subs(\leftarrow \underline{\neg ab(alive,x,[y])}, holds(alive,[y]))$ is the following tree.



The trees relative to $subs(\leftarrow y \neq load, \underline{\neg ab(alive,x_e,[\,])}, holds(alive,[\,]))$ and $subs(\leftarrow x \neq shoot, \underline{\neg ab(alive,x_e,[\,])}, holds(alive,[\,]))$ coincide.

$$\leftarrow \underline{ab(alive, x_e, [\,])}$$

$$\Big\downarrow {\scriptstyle \{x_e/shoot\}\ (d)}$$

$$\leftarrow \underline{holds(loaded, [\,])}$$

$$\Big\downarrow$$

$$\leftarrow \neg ab(loaded, x_e, [\,]), \underline{holds(loaded, [\,])}$$

$$\Big\downarrow$$

$$\diamond$$

Then $(x \neq shoot \vee y \neq load)$ is the full answer substitution for the goal $\leftarrow holds(alive, [x, y])$. $\square$

Theorem 2.1 implies that all variable-free goals terminate when the program is acyclic. We show now that the converse holds, i.e. that a program is acyclic when all variable-free goals terminate. The following notions and lemmas are used.

Consider, for a program P and a goal G, the (super-)tree relative to the set of all SLD-CNF trees with root G, corresponding to all possible selection of literals in goals. We denote this tree by T_G. Let $lnodes(T_G)$ denote the number of nodes of T_G minus the number of resolution steps relative to the selection of negative equality literals. We want to prove that for all θ

$$lnodes(T_G) \geq lnodes(T_{G\theta}) \tag{1}$$

holds, whenever $lnodes(T_G)$ is defined. Let $nodes(T)$ denote the number of nodes of the tree T.

We call a goal G *terminating* if all the paths in T_G are finite, where a *path* of G is a sequence of nodes $G_1, \ldots, G_i, \ldots$ together with the relative sequence of selected literals and input clauses (if any) s.t. for all i, G_{i+1} is either an immediate descendent of G_i in T_G or the root of the tree $subs(G_i)$.

Lemma 2.4 *Let P be a program, θ a substitutiton, G a goal with a positive selected equality literal. If $G\theta$ has resolvent G', then G has resolvent G'' s.t. $G' = G''\rho$ for some substitution ρ.*

Proof. Let $(r = t)$ be the selected literal of G and let $\alpha = mgu(r\theta, t\theta)$. Then $\theta\alpha = \mu\rho$ for some ρ, with $\mu = mgu(r, t)$. The claim follows from $G' = (G - \{(r = t)\})\theta\alpha$ and $G'' = (G - \{(r = t)\})\mu$. $\square$

Lemma 2.5 *Let P be a program, θ a substitution and G a terminating goal. Then $G\theta$ is terminating.*

Proof. By contraposition let $\xi = G_0, G_1, \ldots, G_i, \ldots$ be an infinite path starting at $G_0 = G\theta$. We define inductively the path $\xi' = G'_0, G'_1, \ldots, G'_j, \ldots$ such that, for all j, the selected literal in G'_j is not a constraint and, if it is a negative literal, then G'_{j+1} is not a resolvent of G'_j.

- G'_0 is G_0 with the selected literal and the input clause relative to G_k, where k is the least index greater or equal than 0 s.t. the selected literal of G_k is not a constraint and, if it is a negative literal, then G_{k+1} is not a resolvent of G_k; k exists because ξ is infinite. Set i to $k + 1$ and j to 1.

- Let L be the selected literal of G'_{j-1}. Then two cases arise.

  L is a positive literal. Then define G'_j to be the resolvent of G'_{j-1}. The selected atom and input clause of G'_j are those relative to G_k, where k is the least index greater or equal than i s.t. the selected literal of G_k is not a constraint and, if it is a negative literal, then G_{k+1} is not a resolvent of G_k. Set i to $k+1$ and j to $j+1$.

  $L = \neg A$ is a negative literal. Then define G'_j to be $\leftarrow A$ with input clause that relative to G_i. Set i to $i+1$ and j to $j+1$.

ξ' is by construction an infinite path of $G\theta$ where every resolvent is relative to a positive selected literal. Then it is possible to apply Lemma 1.3 and lift ξ' to a path of G. Then G is not terminating.

$\square$

Lemma 2.6 *Let P be a program and $G = \leftarrow Q$ a terminating goal. Let F_Q be the full answer substitution to Q. Then for all substitutions θ if $G\theta$ is terminating then $F_{Q\theta}$ is equivalent to $F_Q\theta$.*

Proof. By Corollary 2.2 $comp(P) \models \forall \underline{x}(Q \equiv F_Q)$. Then $comp(P) \models \forall \underline{x}(Q\theta \equiv F_Q\theta)$. Since $G\theta$ is terminating, then $F_{Q\theta}$ is defined. By Corollary 2.2 $comp(P) \models \forall \underline{x}(Q\theta \equiv F_{Q\theta})$. Then $F_{Q\theta}$ is equivalent to $F_Q\theta$. $\square$

We have assumed that answers are normalized and negated as described in [Chan88]. Normalized answers do not contain valid conjuncts and the negation of answers does not introduce new subgoals. Thus the following result holds.

Lemma 2.7 *Let P be a program, G a goal with selected literal L and θ a substitution. Then if $L\theta$ is a simple literal then $G\theta$ has less or equal number of resolvents than G and every resolvent of $G\theta$ is an instance of a resolvent of G.*

Proof. If $L\theta$ is a positive simple literal then the claim follows from Lemma 1.3. If $L\theta$ is a negative simple literal then the claim follows by Lemma 2.5 and Lemma 2.6.

$\square$

It remains to consider the case when the selected literal $L\theta$ in the goal $G\theta$ is a negative equality literal. Then it is a valid inequality. But L can be either a valid inequality or a primitive one. In the latter case it cannot be the selected literal of G. Notice however that in this case, since the resolvent of $G\theta$ is $G\theta - \{L\theta\}$, the remaining literals in $G\theta$ are not modified.

From Lemma 2.7 and the previous observation it follows that Property (1) holds. This property allows us to prove the desired claim.

Theorem 2.8 *Let P be a terminating program. Then for some level mapping $|\,|$*

(i) P is acyclic w.r.t. $|\,|$,

(ii) for every goal G, G is bounded w.r.t. $|\,|$ iff G is terminating.

Proof.

Since P is terminating, then by König's Lemma it follows that for every ground atom A the SLD-CNF tree $T_{\leftarrow A}$ is finite. Hence the level mapping that assigns to every ground atom A the number $lnodes(T_{\leftarrow A})$ is well defined. From $lnodes(T_{\leftarrow \neg A}) > lnodes(T_{\leftarrow A})$ it follows that $lnodes(T_{\leftarrow \neg A}) > |\neg A| = |A|$.

(ii1) Consider a terminating goal G. We show that G is bounded by $lnodes(T_G)$. Let $l \in |[G]|$. Then for some ground instance $\leftarrow L_1, \ldots, L_n$ of G and $i \in [1, n]$ we have $l = |L_i|$. Then

$$
\begin{aligned}
& lnodes(T_G) \\
\geq\ & \text{(by Property (1))} \\
& lnodes(T_{(\leftarrow L_1,\ldots,L_n)}) \\
\geq\ & \text{(by construction of } T_{(\leftarrow L_1,\ldots,L_n)}) \\
& lnodes(T_{\leftarrow L_i}) \\
\geq\ & \text{(by the definition of } |\ |) \\
& |L_i| \\
=\ & l.
\end{aligned}
$$

(i) We prove that P is acyclic. Let $A\theta \leftarrow L_1\theta, \ldots, L_n\theta$ a ground instance of a clause in P. We have to show that $|A\theta| > |L_i\theta|$ for all $i \in [1, n]$. Since $A\theta\theta = A\theta$, then θ is a unifier of $A\theta$ and A. Then for some mgu μ of $A\theta$ and A we have $\theta = \mu\theta'$ and $\leftarrow (L_1\mu, \ldots, L_n\mu)$ is a resolvent of $\leftarrow A$. Then

$$
\begin{aligned}
& |A\theta| \\
=\ & \text{(definition of } |\ |) \\
& lnodes(T_{\leftarrow A\theta}) \\
>\ & (T_{\leftarrow(L_1\mu,\ldots,L_n\mu)} \text{ is a subtree of } T_{\leftarrow A\theta}) \\
& lnodes(T_{\leftarrow(L_1\mu,\ldots,L_n\mu)}) \\
\geq\ & \text{(part (ii1), since } L_i\theta \in |[L_1\mu, \ldots, L_n\mu]|) \\
& |L_i\theta|.
\end{aligned}
$$

(ii2) Consider a goal G which is bounded w.r.t. $|\ |$. Then by (i) and Theorem 2.1 G is terminating.

□

3 Procedural Semantics of Acceptable Programs

We consider now procedural semantics of acceptable programs w.r.t. LD-CNF resolution. We show that acceptable programs characterize the class of programs that terminate for all ground goals, for Prolog selection rule.

Theorem 3.1 *Let P be an acceptable program and G a bounded goal. Then every LD-CNF tree for $P \cup \{G\}$ contains only bounded goals and is finite.*

Proof. Let $|\ |$ and I be a level mapping and an interpretation s.t. P is acceptable w.r.t. $|\ |$ and I. Let $G = \leftarrow L_1, \ldots, L_n$, $n \geq 1$ and let L_i be the selected literal, i.e. the leftmost possible literal. We distinguish the following three cases.

Case 1: L_i is a positive simple literal. Then the proof is as that of Lemma 3.7 of Apt and Pedreschi [ApPe90] and is here omitted.

Case 2: L_i is negative, say $\neg A$. Then $subs(G)$ has root $\leftarrow A$ which is obviously bounded and $|[\leftarrow A]|_I$ is smaller or equal than $|[G]|_I$ in the multiset ordering (since $|A| = |\neg A|$); moreover every resolvent G' of G (if any) is bounded and $|[G']|_I$ is smaller than $|[G]|_I$ in the multiset ordering, since it is obtained from G by replacing L_i with a (possibly empty) conjunction of constraints $c_1, \ldots, c_k$ s.t. $I \models L_i \leftarrow c_1, \ldots, c_k$, and s.t. $|c'_i| = 0$ for all ground instance c'_i of c_i, for all $i \in [1, k]$.

Case 3: L_i is a constraint. Then the resolvent G' of G is obtained by removing L_i and applying the relative (if any) substitution. Thus G' is a bounded goal and by Lemma 1.1 $|[G']|_I$ is smaller than $|[G]|_I$ in the multiset ordering.

Now note that there can be only finitely many consecutive selections of negative literals and use the fact that the multiset ordering is well-founded.

□

LD-CNF resolution is complete w.r.t. Clark's semantics for finite trees. So by virtue of Theorem 3.1 the following result holds.

Corollary 3.2 *LD-CNF resolution is sound and complete for bounded goals and acceptable programs w.r.t. Clark completion.*

Theorem 3.1 implies that all variable-free goals terminate when the program is acceptable. To prove the converse, we argue in the same way as for the case of acyclic programs. A property analogous to property (1) is needed. Let T_G denote the LD-CNF tree for $P \cup \{G\}$ and let $lnodes(T_G)$ denote the number of nodes of T_G minus the number of resolution steps relative to the selection of negative equality literals. We want to show that if T_G is finite then for all substitutions θ

$$lnodes(T_G) \geq lnodes(T_{G\theta}). \tag{2}$$

To prove this property we argue in the same way as we did for Property (1). Lemma 2.4 and Lemma 2.6 clearly hold also when LD-CNF resolution is assumed. We call a goal G *left-terminating* if all the paths in T_G are finite. The following is the correspondent of Lemma 2.5 for LD-CNF resolution.

Lemma 3.3 *Let P be a program, θ a substitution and G a left terminating goal. Then $G\theta$ is left terminating.*

Proof. Let $G = \leftarrow L_1, \ldots, L_n$ and let L_i be the selected (leftmost possible) literal of G. Then for all $k \in [1, i-1]$ L_k is a (primitive) negative equality literal. Let $L_{m_1}\theta, \ldots, L_{m_h}\theta$ with $m_1 < \ldots < m_h$, be all the constraints between $L_1\theta$ and $L_i\theta$ which are possible negative equality literals. Then $T_{G\theta}$ is formed by the sequence of nodes $G\theta, G\theta - \{L_{m_1}\theta\}, \ldots, G\theta - \{L_{m_1}\theta, \ldots, L_{m_{h-1}}\theta\}$ followed by the tree $T_{\tau(G\theta)}$, with $\tau(G\theta) = G\theta - \{L_{m_1}\theta, \ldots, L_{m_h}\theta\}$. Notice that the selected leftmost possible literal in $\tau(G\theta)$ is $L_i\theta$. Clearly $T_{G\theta}$ is finite iff $T_{\tau(G\theta)}$ is finite. We prove that

T_G finite implies $T_{\tau(G\theta)}$ finite. We argue by induction on the number $nodes(T_G)$ of nodes of the tree T_G.

Suppose $nodes(T_G) = 1$. Then G either contains only primitive constraints and in this case also $\tau(G\theta)$ contains only primitive constraints, or L_i and hence $L_i\theta$ does not unify with the head of any clause of P. Hence $T_{\tau(G\theta)}$ is finite.

Suppose $nodes(T_G) > 1$. We distinguish the following three cases.

Case 1 L_i is a positive simple literal. Let H be a resolvent of G with input clause C. Then either $\tau(G\theta)$ and C have no resolvent, or by Lemma 1.3 the resolvent H' is an instance of H, i.e. $H' = H\rho$ for some ρ. Since $nodes(T_H) < nodes(T_G)$ and T_H is finite then we can apply the induction hypothesis and conclude that $T_{\tau(H\rho)}$ is finite. Then $T_{\tau(G\theta)}$ is finite.

Case 2 L_i is a negative simple literal. Then $subs(G)$ is finite and $nodes(subs(G)) < nodes(T_G)$. By the induction hypothesis applied to $subs(G)$ it follows that $subs(\tau(G\theta))$ is finite. Moreover by Lemma 2.6 for every resolvent H' of $\tau(G\theta)$ there is a resolvent H of G s.t. $H' = H\theta$. Then $\tau(H') = \tau(H\theta)$.

As $nodes(T_H) < nodes(T_G)$ and T_H is finite, we can apply the induction hypothesis to T_H and conclude that $T_{\tau(H\theta)}$ is finite. Then $T_{\tau(G\theta)}$ is finite.

Case 3 L_i is a constraint. Then by Lemma 2.4 if H is the resolvent of G and H' is the resolvent of $\tau(G\theta)$ we have that $H' = H\rho$ for some ρ. Since $nodes(T_H) < nodes(T_G)$ and T_H is finite, we can apply the induction hypothesis and conclude that $T_{\tau(H')}$ is finite. Hence $T_{\tau(G\theta)}$ is finite.

□

Lemma 3.3 and Lemma 2.6 imply an analogue of Lemma 2.7 for LD-CNF resolution, i.e. if $G = \leftarrow L_1, \ldots, L_n$ and L_1 is a simple literal then for all subsitutions θ $G\theta$ has less or equal number of resolvents than G and every resolvent of $G\theta$ is an instance of a resolvent of G.

Finally when L_1 is a negative equality literal then if it is a primitive inequality it is not selected while $L_1\theta$ can be possibly selected in $G\theta$. However in this case the resolvent of $G\theta$ is $(G - \{L_1\})\theta$, and thus the remaining literals in $G\theta$ are not modified.

Thus property (2) holds. We can now prove the desired claim.

Theorem 3.4 *Let P be a left terminating program. Then for some level mapping $|\ |$ and a model I of $comp(P)$*

(i) P is acceptable w.r.t. $|\ |$ and I,

(ii) for every goal G, G is bounded w.r.t. $|\ |$ and I iff all LD-CNF derivations of $P \cup \{G\}$ are finite .

Proof.

Since P is left terminating, then by König's Lemma it follows that for every ground atom A the SLD-CNF tree $T_{\leftarrow A}$ is finite. Hence the level mapping that assigns to every ground atom A the number $lnodes(T_{\leftarrow A})$ is well defined. From $lnodes(T_{\leftarrow\neg A}) > lnodes(T_{\leftarrow A})$ it follows that $lnodes(T_{\leftarrow\neg A}) > |\neg A| = |A|$.

Next, choose $I = \{A \in B_P \mid \text{there is an LD-CNF refutation of } P \cup \{\leftarrow A\}\}$. The proof that I is a model of $comp(P)$ is contained in the proof of Theorem 3.4 of Apt and Pedreschi [ApPe91, page 12] and is here omitted.

(ii1) Consider a left terminating goal G. We show that G is bounded by $lnodes(T_G)$. Let $l \in |[G]|$. Then for some ground instance $\leftarrow L_1, \ldots, L_n$ of G and $i \in [1, \overline{n}]$ with $\overline{n} = min(\{n\} \cup \{i \in [1,n] \mid I \not\models L_i\})$, we have $l = |L_i|$. Then

$$
\begin{aligned}
& lnodes(T_G) \\
\geq\ & \text{(Property (2))} \\
& lnodes(T_{(\leftarrow L_1, \ldots, L_n)}) \\
\geq\ & \text{(by construction of } T_{(\leftarrow L_1, \ldots, L_n)}) \\
& lnodes(T_{\leftarrow L_1, \ldots, L_{\overline{n}}}) \\
\geq\ & \text{(by construction of } T_{(\leftarrow L_1, \ldots, L_n)}) \\
& lnodes(T_{\leftarrow L_i, \ldots, L_{\overline{n}}}) \\
\geq\ & \text{(by construction of } T_{(\leftarrow L_1, \ldots, L_n)}) \\
& lnodes(T_{\leftarrow L_i}) \\
\geq\ & \text{(by definition of } |\ |) \\
& |L_i| \\
=\ & l.
\end{aligned}
$$

(i) We prove that P is acceptable. Let $A\theta \leftarrow L_1\theta, \ldots, L_n\theta$ a ground instance of a clause in P. We have to show that $|A\theta| > |L_i\theta|$ for all $i \in [1, \overline{n}]$, with $\overline{n} = min(\{n\} \cup \{i \in [1,n] \mid I \not\models L_i\})$. Since $A\theta\theta = A\theta$, then θ is a unifier of $A\theta$ and A. Then for some mgu μ of $A\theta$ and A we have $\theta = \mu\theta'$ and $\leftarrow (L_1\mu, \ldots, L_n\mu)$ is a resolvent of $\leftarrow A$. Then

$$
\begin{aligned}
& |A\theta| \\
=\ & \text{(definition of } |\ |) \\
& lnodes(T_{\leftarrow A\theta}) \\
>\ & (T_{\leftarrow (L_1\mu, \ldots, L_n\mu)} \text{ is a subtree of } T_{\leftarrow A\theta}) \\
& lnodes(T_{\leftarrow (L_1\mu, \ldots, L_n\mu)}) \\
\geq\ & \text{(part (ii1), since } L_i\theta \in |[L_1\mu, \ldots, L_n\mu]|) \\
& |L_i\theta|.
\end{aligned}
$$

(ii2) Consider a goal G which is bounded w.r.t. $|\ |$. Then by (i) and Theorem 3.1 G is left terminating.

□

4 Partial Correctness of General Programs

Our method to prove partial correctness of logic programs comes from imperative programming (see for instance [Hoare69]). In that setting formulas of the form $\{\varphi\}\Gamma\{\phi\}$ are considered with the meaning that if the assertion φ is true before the initiation of the program Γ then the assertion ϕ will be true on its completion. To prove the correctness of such a formula the program is decorated with invariant assertions that describe the state of the computation when the control reaches the corresponding program point. We consider here Hoare-like triples of the form $\{pre(L)\}L\{post(L)\}$ to express the partial correctness of a literal L together with a pre- and a post-condition with respect to a general program P. Both SLD-CNF (i.e. arbitrary selection rule) and LD-CNF resolution (i.e. Prolog selection rule) are considered.

In SLD-CNF (resp. LD-CNF) resolution a c.a.s. is a substitution together with a set of primitive constraints and it can be represented by a *simple equality formula* (see Section 1). We call *characteristic assertion* of a substitution α the equality formula associated with α, denoted by $\mathcal{A}_\alpha$.

The intended meaning of $\{pre(L)\}L\{post(L)\}$ is that if the assertion $pre(L)$ is true before the execution of the goal $\leftarrow L$ in the program P, then the assertion $post(L)$ is *implied* by the characteristic assertion of every SLD-CNF (resp. LD-CNF) c.a.s. of $\leftarrow L$.

We introduce a proof-method to prove the correctness of $\{pre(L)\}L\{post(L)\}$ w.r.t. an acyclic program and prove its soundness w.r.t. SLD-CNF resolution. Similarly we introduce a proof-method to prove the correctness of $\{pre(L)\}L\{post(L)\}$ w.r.t. an acceptable program and we prove its soundness w.r.t. LD-CNF resolution. Finally we show how these methods can be extended to general programs.

We consider *monotonic* assertions, i.e. assertions p s.t. $p\alpha$ true implies $p\alpha\beta$ true for all substitutions β. Thus to describe the form of an argument x of a goal during its execution we can use the assertion $ground(x)$ or $\neg var(x)$, but not $var(x)$, since this predicate is not monotonic. Notice that $(s \neq t)\alpha$ is true if all its ground instances are true, i.e. if it is a valid inequality. Thus negative equality literals are monotonic predicates. Moreover we allow a specification to contain assertions that are finite representations of an infinite (countable) number of disjuncts or conjuncts. For instance the assertion $\exists n \exists X(L = [X(1), \ldots, X(n)])$ is a finite representation of the infinite assertion $(L = [\,] \vee L = [X(1)] \vee L = [X(1), X(2)] \vee \ldots)$, which says that L is a list. We write for simplicity this assertion as $\exists x_1, \ldots, x_n(L = [x_1, \ldots, x_n])$. We call a specification $\{pre(L)\}L\{post(L)\}$ positive (resp. negative) if L is a simple positive (resp. negative) literal; we call $\{pre(L)\}L\{post(L)\}$ constraint specification if L is a constraint.

Definitions and results concerning constraint and negative specifications are analogous for both acyclic and acceptable programs. The correctness of a constraint specification does not depend on the considered program.

Definition 4.1 The correctness of a constraint specification $\{pre(L)\}L\{post(L)\}$ is defined as follows:

- if L is a *negative equality literal* then the implication $pre(L) \rightarrow post(L)$ holds;
- if L is a *positive equality literal* then the implication $(pre(L) \wedge L) \rightarrow post(L)$ holds.

For negative specifications $\{pre(L)\}L\{post(L)\}$ we need to know all the c.a.s.'s of $P \cup \{\leftarrow L\}$ starting with preconditon $pre(L)$ true.

Definition 4.2 (Strongest Postcondition) Let L be a negative simple literal, P a program, p an assertion. We call *strongest postcondition* of L w.r.t. p and P, denoted by $sp_P.L.p$, the assertion

$$\bigvee_{\alpha,\beta} (\mathcal{A}_\alpha \wedge \mathcal{A}_\beta)$$

for all α and β s.t. $p\alpha$ is true and β is a c.a.s. for $P \cup \{\leftarrow L\alpha\}$ w.r.t. SLD-CNF (resp. LD-CNF) resolution.

We write for simplicity $sp.L.p$ instead of $sp_P.L.p$ when ambiguity does not arise.

Definition 4.3 (Correctness of Negative Specifications) We say that a *negative specification* $\{pre(L)\}L\{post(L)\}$ is *correct w.r.t. a program* P if the implication

$$sp.L.pre(L) \rightarrow post(L)$$

holds.

Notice that the definition of correctness of a negative specification is given in semantic terms, i.e. it refers to substitutions and c.a.s.'s. However for acyclic (resp. acceptable) programs the soundness and completeness of SLD-CNF (resp. LD-CNF) resolution for bounded goals w.r.t. Clark's completion implies that if B is a bounded atom and F_B is the full answer substitution to B then $comp(P) \models B \leftrightarrow F_B$. This property can be used to provide a criterion to find an assertion equivalent to $sp.L.p$ w.r.t. Definition 4.2. We denote by $\underline{x}$ (resp. $\underline{t}$) a sequence of variables (resp. of terms).

Theorem 4.4 *Let A be an atom, P a program, p an assertion. If $p \rightarrow (A = \exists \underline{x} B)$ with B bounded atom, whose free-variables occur in A, then $sp.\neg A.p$ is equivalent, w.r.t. Definition 4.2, to $(p \wedge \exists \underline{x}(A = B \wedge \phi_{\neg B}))$, with $\phi_{\neg B}$ equality formula equivalent to $\neg B$ w.r.t. $comp(P)$.*

Proof. If p is equivalent to $FALSE$ then obviously $sp.\neg A.p$ is equivalent to $FALSE$. Otherwise: Let α and β s.t. $p\alpha$ is true and β is a c.a.s. for $P \cup \{\leftarrow \neg A\alpha\}$. We have to show that

$$(\mathcal{A}_\alpha \wedge \mathcal{A}_\beta) \rightarrow (p \wedge \exists \underline{x}(A = B \wedge \phi_{\neg B})).$$

By hypothesis there exists a substitution $\{\underline{x}/\underline{t}\}$ s.t. $A\alpha = B\alpha\{\underline{x}/\underline{t}\}$ holds and $\leftarrow \neg B\alpha\{\underline{x}/\underline{t}\}$ is bounded. Then β is a c.a.s. for $P \cup \{\leftarrow \neg B\alpha\{\underline{x}/\underline{t}\}\}$. By the soundness of SLD-CNF (resp. LD-CNF) resolution for bounded goals and acyclic (resp. acceptable) programs w.r.t. $comp(P)$ it follows that the implication $(\mathcal{A}_\beta \rightarrow \phi_{\neg B}\alpha\{\underline{x}/\underline{t}\})$ holds. Then, as p is monotonic and $p\alpha$ is true, we obtain that the implication $(\mathcal{A}_\alpha \wedge \mathcal{A}_\beta) \rightarrow (p \wedge \exists \underline{x}(A = B \wedge \phi_{\neg B}))$ also holds.

Conversely let α be s.t. $(p \wedge \exists \underline{x}(A = B \wedge \phi_{\neg B}))\alpha$ is true. By the hypothesis $A\alpha = B\alpha\{\underline{x}/\underline{t}\}$ holds for some substitution $\{\underline{x}/\underline{t}\}$. Then it is sufficient to show that ϵ is a c.a.s. for $P \cup \{\leftarrow \neg B\alpha\{\underline{x}/\underline{t}\}\}$. By hypothesis $\leftarrow \neg B$ is bounded: then by Lemma 1.1 it follows that $\leftarrow \neg B\alpha\{\underline{x}/\underline{t}\}$ is bounded. As $\phi_{\neg B}\alpha\{\underline{x}/\underline{t}\}$ is true, it follows from the completeness of SLD-CNF (resp. LD-CNF) resolution for bounded goals and acyclic (resp. acceptable) programs w.r.t. $comp(P)$ and from $A\alpha = B\alpha\{\underline{x}/\underline{t}\}$ it follows that $\mathcal{A}_\alpha \rightarrow sp.\neg A.p$. □

Notice that in the proof of Theorem 4.4 both the soundness and completeness of SLD-CNF (resp. LD-CNF) resolution w.r.t. the completion of the program are used.

Note 4.5 Theorem 4.4 follows from Corollary 2.2 (resp. Corollary 3.2) if the considered assertion p is $TRUE$. In fact in this case Theorem 4.4 says that if $\neg A$ is bounded then $\forall(\neg A \equiv (\exists A_1 \vee \ldots \vee \exists A_n))$ with $A_1, \ldots, A_n$ all the c.a.s.'s for $P \cup \{\leftarrow \neg A\}$. However in general Theorem 4.4 allows to characterize the c.a.s.'s of the class of negative atomic goals specified by a negative atom $\neg A$ and a precondition p, by considering a bounded negative atomic goal which is equivalent to $\leftarrow \neg A$ w.r.t. p and by translating it into an equality formula equivalent w.r.t. the completion of the program.

Note 4.6 Consider negation as failure. If L is a ground literal and P is an acyclic (resp. acceptable) program then $comp(P) \models (L \leftrightarrow \phi_L)$ where ϕ_L is either $TRUE$ or $FALSE$. Then if p implies that L is ground and L does not flounder, then $sp.L.p$ is either p or $FALSE$ and Theorem 4.4 gives a sufficient criterion to decide which of them.

Example 4.7 Consider the program P

```
elem(X,[X|L]) ←
elem(X,[H|L]) ←  elem(X,L)
```

It is easy to check that P is acyclic by choosing as level mapping $|elem(s,t)| = |t|$ where if t is a list then $|t|$ is its size, otherwise $|t|$ is zero. We calculate

$$sp.\neg elem(x,l).\{list(l)\}.$$

We have that

$$list(l) \rightarrow (\exists x_1,\ldots,x_n(elem(x,l) = elem(x,[x_1,\ldots,x_n])))$$

holds. Moreover for an arbitrary n $\neg elem(x,[x_1,\ldots,x_n])$ is bounded by n itself and

$$comp(P) \models (\neg elem(x,[x_1,\ldots,x_n]) \leftrightarrow (x \neq x_1 \wedge \ldots \wedge x \neq x_n)).$$

Then by Theorem 4.4 $sp.\neg elem(x,l).\{list(l)\}$ is equivalent (w.r.t. Definition 4.2) to

$$(list(l) \wedge \exists x_1,\ldots,x_n(elem(x,l) = elem(x,[x_1,\ldots,x_n]) \wedge x \neq x_1 \wedge \ldots \wedge x \neq x_n)).$$

□

Let us now consider positive specifications. To prove the partial correctness of a positive specification we argue as in [CoMa91], [Dr91]. The program is decorated with assertions, a pre- and a postcondition for every literal in the body of a clause. These assertions are proved to be global invariants and the specification is shown to *match* every asserted clause of the program. Informally a positive specification $\{pre(A)\}A\{post(A)\}$ matches an asserted clause if

- the precondition of the clause can be obtained from $pre(A)$ and the unification of A with the head H of the clause,

- $post(A)$ can be obtained from $pre(A)$, the postcondition of the clause and the unification of A with H.

Since we consider monotonic assertions, the information regarding the unification of A with H can be expressed by the equality assertion $A = H$.

The selection rule is relevant in the method. If we consider an arbitrary selection rule then the precondition (resp. postcondition) of an asserted clause is the conjunction of the preconditions (resp. postconditions) of the literals in the body of the clause. If we consider Prolog selection rule we have that the literals in the body of a clause are ordered from left to right. Then the precondition (resp. postcondition) of an asserted clause is the precondition (resp. postcondition) of the first (resp. last) literal in the body of the clause. Moreover with Prolog selection rule the postcondition of a literal in the body of a clause coincides with the precondition of the next literal.

Definition 4.8 (Asserted Clause) An *asserted clause* $\mathcal{AC}$ is a clause C together with a specification $\{pre(L)\}L\{post(L)\}$ associated to each of its body literals L.

Definition 4.9 (Asserted Program) An *asserted program* $\mathcal{AP}$ consists of a set of asserted clauses $\mathcal{AC}$, one for every clause C of P.

Definition 4.10 (Matching) We say that the specification $\{pre(L)\}L\{post(L)\}$ *matches* the asserted program $\mathcal{A}P$ if the following conditions hold:

- If L is a constraint then $\{pre(L)\}L\{post(L)\}$ is correct.
- If L is a positive simple literal then

  1) for every disjoint with $\{pre(L)\}L\{post(L)\}$ variant $H' \leftarrow$ of a unit clause of P s.t. L and H' unify the following implication holds:

  $$(pre(L) \wedge L = H') \rightarrow post(L);$$

  2) for every disjoint with $\{pre(L)\}L\{post(L)\}$ variant $\mathcal{A}C'$ of a non-unit asserted clause of $\mathcal{A}P$ s.t. L and the head H' of C' unify the following implications hold:

  $$(pre(L) \wedge L = H') \rightarrow pre(\mathcal{A}C')$$
  $$(pre(L) \wedge post(\mathcal{A}C') \wedge L = H') \rightarrow post(L).$$

- If L is a negative simple literal then $\{pre(L)\}L\{post(L)\}$ is correct w.r.t. P.

We call $pre(\mathcal{A}C)$ and $post(\mathcal{A}C)$ respectively pre- and postcondition of $\mathcal{A}C$. Since they will depend on the considered selection rule, two different definitions will be given in the follow respectively for acyclic and acceptable programs.

Definition 4.11 (Partial Correct Asserted Program) We say that the asserted program $\mathcal{A}P$ is *partially correct* if its specifications match $\mathcal{A}P$.

Definition 4.12 (Partial Correct Positive Specification) We say that a positive specification $\{pre(L)\}L\{post(L)\}$ is partially correct w.r.t. a program P if there exists a partially correct asserted program $\mathcal{A}P$ s.t. $\{pre(L)\}L\{post(L)\}$ matches $\mathcal{A}P$.

We denote by $\underline{L}_{i,n}$ the sequence of literals $L_i, \ldots, L_n$ and by $pre(\underline{L}_{i,n})$ (resp. $post(\underline{L}_{i,n})$) the assertion $pre(L_i) \wedge \ldots \wedge pre(L_n)$ (resp. $post(L_i) \wedge \ldots \wedge post(L_n)$). We write simply $\underline{L}_n$ for $\underline{L}_{1,n}$.

The following generalization of partial correctness to specifications that contain sequences of extended literals is useful.

Definition 4.13 We say that a general specification $\{pre(\underline{L}_n)\}\underline{L}_n\{post(\underline{L}_n)\}$ is partially correct w.r.t. the program P if there exists a partial correct asserted program $\mathcal{A}P$ s.t. for every $i \in [1, n]$ one of the following conditions holds:

1) If L_i is a constraint then $\{pre(L_i)\}L_i\{post(L_i)\}$ is correct,

2) If L_i is a negative literal then $\{pre(L_i)\}L_i\{post(L_i)\}$ is correct w.r.t. P,

3) If L_i is a positive literal then $\{pre(L_i)\}L_i\{post(L_i)\}$ matches $\mathcal{A}P$.

4.1 Partial Correctness of Acceptable Programs

In this section whenever we write program we assume it is acceptable and whenever we write c.a.s. we assume it is a c.a.s. with respect to LD-CNF resolution.

Definition 4.14 Let $\mathcal{A}C$ be an asserted clause and let $L_1, \ldots, L_n$ be the body of C. Then $pre(\mathcal{A}C) = \{pre(L_1)\}$ and $post(\mathcal{A}C) = \{post(L_n)\}$. Moreover $\forall i \in [1, n-1]$ $(post(L_i) = pre(L_{i+1}))$.

Now Definition 4.12 together with Theorem 4.4 provide a method to prove the partial correctness of an acceptable program. To show that this method is correct we consider general specifications. We denote by $\{R_0\}L_1\{R_1\}\ldots\{R_{n-1}\}L_n\{R_n\}$ a general specification. The following property is useful.

Lemma 4.15 *Let p, q, p', q' be assertions, L an extended literal and P a general program. If the implications $p \rightarrow p'$ and $q' \rightarrow q$ hold and $\{p'\}L\{q'\}$ is correct w.r.t. P then $\{p\}L\{q\}$ is correct w.r.t. P.*

Proof. The case L constraint is immediate.

If L is a negative simple literal then by Definition 4.2 it follows that $p \rightarrow p'$ implies $sp.L.p \rightarrow sp.L.p'$. Moreover by Definition 4.3 it follows that $q' \rightarrow q$ implies $sp.L.p' \rightarrow q$. Then $\{p\}L\{q\}$ is correct.

If L is a positive simple literal then let $\mathcal{A}P$ be a correct asserted program s.t. $\{p'\}L\{q'\}$ matches $\mathcal{A}P$. We show that $\{p\}L\{q\}$ matches $\mathcal{A}P$. For every asserted clause $\mathcal{A}C$ in $\mathcal{A}P$ we have that

$$(p' \wedge H = L) \rightarrow pre(\mathcal{A}C) \tag{3}$$

and

$$(p' \wedge post(\mathcal{A}C) \wedge H = L) \rightarrow q', \tag{4}$$

where H is the head of $\mathcal{A}C$. From $p \rightarrow p'$ and (3) it follows that $(p \wedge H = L) \rightarrow pre(\mathcal{A}C)$ and from $p \rightarrow p'$, $q' \rightarrow q$ and (3) it follows that $(p \wedge post(\mathcal{A}C) \wedge H = L) \rightarrow q$. Then $\{p\}L\{q\}$ matches $\mathcal{A}P$.

□

Theorem 4.16 (Soundness I) *If $\{R_0\}L_1\{R_1\}\ldots\{R_{n-1}\}L_n\{R_n\}$ is correct w.r.t. the program P then for all α, $\mathcal{A}$ s.t. $R_0\alpha$ is true and $\mathcal{A}$ is an answer to $\underline{L}_n\alpha$ relative to a LD-CNF refutation of $P \cup \{\leftarrow \underline{L}_n\alpha\}$ the following implication holds:*

$$(\mathcal{A}_\alpha \wedge \mathcal{A}) \rightarrow R_n.$$

Proof. By Definition 4.13 there exists a correct asserted program $\mathcal{A}P$ s.t. conditions 1), 2) and 3) hold. Let α s.t. $R_0\alpha$ is true. Let ξ be a refutation of $P \cup \{\leftarrow \underline{L}_n\alpha\}$ with sequence of computed substitutions $\beta_1, \ldots, \beta_k$ and conjunction of primitive inequalities c. We proceed by induction on the number l of resolvents in ξ.

Suppose $l = 0$. Then L_i is a primitive inequality for every $i \in [1, n]$ and $c = L_1 \wedge \ldots \wedge L_n$. By Definition 4.1 $R_{i-1} \rightarrow R_i$ for all $i \in [1, n]$. Then from $R_0\alpha$ true we conclude $(\mathcal{A}_\alpha \wedge c) \rightarrow R_n$.

Suppose $l > 0$. Let $L_i\alpha$ be the selected literal. Then $L_i\alpha$ is the leftmost possible literal, i.e. for all $j \in [1, i-1]$ L_j is a primitive inequality. Then for all $j \in [1, i-1]$

$$R_{j-1} \rightarrow R_j. \tag{5}$$

We distinguish the following three cases.

Case 1 $L_i\alpha$ is a negative simple literal. Then $\beta_1 = \epsilon$. Let $G = \leftarrow \underline{L}_{i-1}\alpha, c_1, \ldots, c_k, \underline{L}_{i+1,n}\alpha$ be its resolvent in ξ, where $c_1, \ldots, c_k$ are equality literals. Consider the general specification

$$\{\mathcal{I}\}L_1\{\mathcal{I}\}\ldots\{\mathcal{I}\}L_{i-1}\{\mathcal{I}\}c_1\{\mathcal{I}^{c_1}\}\ldots\{\mathcal{I}^{c_{k-1}}\}c_k\{R_i\}, \tag{6}$$

where $\mathcal{I} = \mathcal{A}_\alpha$ and for all $j \in [0, k]$ $\mathcal{I}^{c_j}$ is defined as follows .

$$\mathcal{I}^{c_0} = \mathcal{I}$$

and for $j > 0$

$$\mathcal{I}^{c_j} = \begin{cases} \mathcal{I}^{c_{j-1}} & \text{if } c_j \text{ is a negative equality literal,} \\ \mathcal{I}^{c_{j-1}} \wedge c_j & \text{if } c_j \text{ is a positive equality literal.} \end{cases}$$

From (5) it follows that $R_{i-1}\alpha$ is true. Let $c'_1, \ldots, c'_h$ be the c.a.s. to $\leftarrow c_1, \ldots, c_k$. Then $c'_1, \ldots, c'_h$ is a c.a.s. for $\leftarrow L_i\alpha$ and

$$c_1, \ldots, c_k \leftrightarrow c'_1, \ldots, c'_h \tag{7}$$

holds (w.r.t. *comp*(P)). From $\{R_{i-1}\}L_i\{R_i\}$ partially correct we have that

$$(\mathcal{A}_\alpha \wedge c'_1 \wedge \ldots \wedge c'_h) \rightarrow R_i \text{ holds.} \tag{8}$$

Then from (7) and (8) it follows that $\{\mathcal{I}\}c_k\{R_i\}$ is correct. Hence the specification (6) is partially correct. Moreover the specification $\{R_i\}\underline{L}_{i+1,n}\{R_n\}$ is partially correct by hypothesis. Hence

$$\{\mathcal{I}\}L_1\{\mathcal{I}\}\ldots\{\mathcal{I}\}L_{i-1}\{\mathcal{I}\}c_1\{\mathcal{I}^{c_1}\}\ldots\{\mathcal{I}^{c_{k-1}}\}c_k\{R_i\}\underline{L}_{i+1,n}\{R_n\}$$

is partially correct.

From $\mathcal{I}\alpha\beta_1$ true we can apply the induction hypothesis to the subrefutation of ξ starting at G. We obtain

$$(\mathcal{A}_{\alpha\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow R_n.$$

Then the claim follows as $\mathcal{A}_{\alpha\beta_1}$ is implied by $\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1}$.

Case 2 $L_i\alpha$ is a constraint. Let $G = \leftarrow (\underline{L}_{i-1}, \underline{L}_{i+1,n})\alpha\beta_1$ be the resolvent. We distinguish the following two cases.

Case a) $L_i\alpha$ is a negative equality literal. Then $\beta_1 = \epsilon$. From $\{R_{i-1}\}L_i\{R_i\}$ correct it follows that $R_{i-1} \rightarrow R_i$ holds. Then the specification

$$\{R_0\}L_1\{R_1\}\ldots\{R_{i-1}\}L_{i-1}\{R_i\}L_{i+1}\{R_{i+1}\}\ldots L_n\{R_n\}$$

is partially correct.

Moreover $R_0\alpha\beta_1$ is true.

Case b) $L_i\alpha$ is a positive equality literal, say $s = t$. Then $\beta_1 = mgu(s,t)$. From $\{R_{i-1}\}L_i\{R_i\}$ correct it follows that $(R_{i-1} \wedge L_i) \rightarrow R_i$ holds. Then the specification

$$\{R_0 \wedge L_i\}L_1\{R_1 \wedge L_i\}\ldots\{R_{i-1} \wedge L_i\}L_{i-1}\{R_i\}. \tag{9}$$

is partially correct. Moreover the specification $\{R_i\}\underline{L}_{i+1,n}\{R_n\}$ is partially correct by hypothesis. Hence

$$\{R_0 \wedge L_i\}L_1\{R_1 \wedge L_i\}\ldots\{R_{i-1} \wedge L_i\}L_{i-1}\{R_i\}L_{i+1}\{R_{i+1}\}\ldots L_n\{R_n\}$$

is partially correct.

Moreover $(R_0 \wedge L_i)\alpha\beta_1$ is true.

In both cases we can apply the induction hypothesis to the subrefutation of ξ starting at G. We obtain

$$(\mathcal{A}_{\alpha\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow R_n.$$

Then the claim follows as $\mathcal{A}_{\alpha\beta_1}$ is implied by $\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1}$.

Case 3 $L_i\alpha$ is a simple positive literal. Let $C = H \leftarrow \underline{B}_m$ be the input clause. Then $\beta_1 = mgu(L_i, H)$. We distinguish the following two cases.

Case a) $m = 0$, i.e. C is a unit clause. Then the resolvent is $G = \leftarrow (\underline{L}_{i-1}, \underline{L}_{i+1,n})\alpha\beta_1$. From $\{R_{i-1}\}L_i\{R_i\}$ matches $\mathcal{A}P$ it follows that $(R_{i-1} \wedge H = L_i) \rightarrow R_i$.

Then the specification

$$\{R_0 \wedge H = L_i\}L_1\{R_1 \wedge H = L_i\}\ldots\{R_{i-1} \wedge H = L_i\}L_{i-1}\{R_i\} \tag{10}$$

is correct. Moreover the specification $\{R_i\}\underline{L}_{i+1,n}\{R_n\}$ is partially correct by hypothesis. Hence

$$\{R_0 \wedge H = L_i\}L_1\{R_1 \wedge H = L_i\}\ldots\{R_{i-1} \wedge H = L_i\}L_{i-1}\{R_i\}L_{i+1}\{R_{i+1}\}\ldots L_n\{R_n\}$$

is partially correct. Then from $(R_0 \wedge H = L_i)\alpha\beta_1$ true we can apply the induction hypothesis to the subrefutation of ξ starting at G. We obtain

$$(\mathcal{A}_{\alpha\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow R_n.$$

Then the claim follows as $\mathcal{A}_{\alpha\beta_1}$ is implied by $\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1}$.

Case b) $m > 0$. Then the resolvent is $G = \leftarrow (\underline{L}_{i-1}, \underline{B}_m, \underline{L}_{i+1,n})\alpha\beta_1$. As $\{R_{i-1}\}L_i\{R_i\}$ matches $\mathcal{A}P$, it follows that $(R_{i-1} \wedge H = L_i) \rightarrow pre(\mathcal{A}C)$ and $(R_{i-1} \wedge H = L_i \wedge post(\mathcal{A}C)) \rightarrow R_i$ hold.

Then, as $\mathcal{A}P$ is correct, from Lemma 4.15 we have that the specification

$$\{R_{i-1} \wedge H = L_i\}\underline{B}_m\{R_i\} \tag{11}$$

is partially correct. Moreover from (5) and L_j negative equality literal for every $j \in [1, i-1]$, it follows that $\{R_0 \wedge H = L_i\}\underline{L}_{i-1}\{R_{i-1} \wedge H = L_i\}$ is correct. Hence

$$\{R_0 \wedge H = L_i\}\underline{L}_{i-1}\{R_{i-1} \wedge H = L_i\}\underline{B}_m\{R_i\}\underline{L}_{i+1,n}\{R_n\}$$

is correct. Then from $(R_0 \wedge H = L_i)\alpha\beta_1$ true we can apply the induction hypothesis to the subrefutation of ξ starting at G. We obtain

$$(\mathcal{A}_{\alpha\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow R_n.$$

Then the claim follows as $\mathcal{A}_{\alpha\beta_1}$ is implied by $\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1}$. $\square$

Example 4.17 This example illustrates how our method can be used to show that a program does not flounder. Consider the following asserted version $\mathcal{A}GAME$ of the program $GAME$.

$$
\begin{array}{lll}
\texttt{win(X)} & \leftarrow & \{TRUE\}\ \texttt{move(X, Y)}\ \{(X,Y)\in\mathcal{G}\}\ \neg\ \texttt{win(Y)}\ \{(X,Y)\in\mathcal{G}\} \\
\texttt{move(a, b)} & \leftarrow & \texttt{for}\ \ (a,b)\in\mathcal{G}
\end{array}
$$

where $\mathcal{G}$ is an acyclic finite graph, whose domain $dom(\mathcal{G})$ contains only ground terms.

We show that for every term t $GAME \cup \{\leftarrow win(t)\}$ does not flounder.

It has been proven in [ApPe91] that $GAME$ is acceptable (but not acyclic). Then to show that $GAME \cup \{\leftarrow win(t)\}$ does not flounder it is sufficient to prove that

a) $\mathcal{A}GAME$ is correct and

b) $\{TRUE\}$ $win(t)$ $\{TRUE\}$ matches $\mathcal{A}GAME$.

In fact by the soundness of our method w.r.t. LD-CNF resolution (Theorem 4.16) it follows that every time the negative goal $\leftarrow \neg win(Y)$ is executed its argument Y is ground.

It is immediate to check that $\{TRUE\}$ $win(t)$ $\{TRUE\}$ matches $\mathcal{A}GAME$, since the precondition of the first asserted clause of $\mathcal{A}GAME$ and the postcondition of the specification are both $TRUE$.

To prove the correctness of $\mathcal{A}GAME$ reduces to the following implications.

$$(move(X,Y) = move(a,b)) \rightarrow (X,Y) \in \mathcal{G},$$

$$sp.\neg win(Y).\{(X,Y)\in\mathcal{G}\} \rightarrow (X,Y)\in\mathcal{G}.$$

The first implication holds by construction. To show that the second implication is also valid we argue as follows. The assertion $(X,Y)\in\mathcal{G}$ implies that Y is in the domain of $\mathcal{G}$. Then

$$(X,Y)\in\mathcal{G} \rightarrow \exists a \in dom(\mathcal{G})(win(Y) = win(a)).$$

In [ApPe91, page 18] it is proven that $win(a)$ is a bounded atom. Then by Theorem 4.4

$$sp.\neg win(Y).\{(X,Y)\in\mathcal{G}\} \equiv ((X,Y)\in\mathcal{G} \wedge \exists a \in dom(\mathcal{G})(win(Y) = win(a) \wedge \phi_{\neg win(a)}),$$

where $\phi_{\neg win(a)}$ is the equality formula equivalent to $\neg win(a)$ w.r.t. $comp(GAME)$. This assertion clearly implies $(X,Y)\in\mathcal{G}$.

Hence $\mathcal{A}GAME$ is correct and $\{TRUE\}$ $win(t)$ $\{TRUE\}$ matches $\mathcal{A}GAME$.

4.2 Partial Correctness of Acyclic Programs

In this section whenever we write program we assume it is acyclic and whenever we write c.a.s. we assume it is a c.a.s. with respect to SLD-CNF resolution. The following definitions are useful.

Definition 4.18 Let $\mathcal{A}C$ be an asserted clause and let $L_1,\ldots,L_n$ be the body of C. Then $pre(\mathcal{A}C) = \{pre(L_1)\wedge\ldots\wedge pre(L_n)\}$ and $post(\mathcal{A}C) = \{post(L_1)\wedge\ldots\wedge post(L_n)\}$.

Now Definition 4.12 together with Theorem 4.4 provide a method to prove the partial correctness of an acyclic program. To show that this method is correct we prove the following stronger claim.

Theorem 4.19 (Soundness II) *If* $\{pre(\underline{L}_n)\}\underline{L}_n\{post(\underline{L}_n)\}$ *is partial correct w.r.t. the program* P *then for all* α, $\mathcal{A}$ *s.t.* $pre(\underline{L}_n)\alpha$ *is true and* $\mathcal{A}$ *is an answer to* $\underline{L}_n\alpha$ *relative to a SLD-CNF refutation of* $P \cup \{\leftarrow \underline{L}_n\alpha\}$ *the following implication holds:*

$$(\mathcal{A}_\alpha \wedge \mathcal{A}) \rightarrow post(\underline{L}_n).$$

Proof. By Definition 4.13 there exists a partial correct asserted program $\mathcal{A}P$ s.t. conditions 1), 2) and 3) hold. Let α s.t. $pre(\underline{L}_n)\alpha$ is true. Let ξ be a refutation of $P \cup \{\leftarrow \underline{L}_n\alpha\}$ with sequence of computed substitutions $\beta_1, \ldots, \beta_k$ and conjunction of primitive inequalities c. We proceed by induction on the number l of resolvents of ξ. The base case $l = 0$ is immediate because it implies that L_i is a primitive inequality for every $i \in [1, n]$ and hence, by Definition 4.1 $pre(L_i) \rightarrow post(L_i)$ holds.

Assume $l > 0$. Let $L_i\alpha$ be the selected extended literal. We distinguish the following three cases.

Case 1 $L_i\alpha$ is a negative simple literal. In this case $\beta_1 = \epsilon$. Let $G = \leftarrow \underline{L}_{i-1}\alpha, c_1, \ldots, c_k, \underline{L}_{i+1,n}\alpha$ be its resolvent in ξ. For every $i \in [1, k]$ choose $pre(c_i) = post(c_i) = TRUE$ if c_i is a negative equality literal, $pre(c_i) = TRUE$ and $post(c_i) = c_i$ if c_i is a positive equality literal. Then $\{pre(c_i)\}c_i\{post(c_i)\}$ is correct for every $i \in [1, k]$. As $(pre(\underline{L}_{i-1}) \wedge pre(c_1) \wedge \ldots \wedge pre(c_k) \wedge pre(\underline{L}_{i+1,n}))\alpha\beta_1$ is true then, from the induction hypothesis applied to the subrefutation of ξ starting at G, we have that

$$(\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow (post(\underline{L}_{i-1}) \wedge post(c_1) \ldots \wedge post(c_k) \wedge post(\underline{L}_{i+1,n})). \tag{12}$$

Let $c'_1, \ldots, c'_h$ be the c.a.s. to $\leftarrow c_1, \ldots, c_k$. Then $c'_1, \ldots, c'_h$ is a c.a.s. for $\leftarrow L_i\alpha$. Hence

$$(\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow (\mathcal{A}_\alpha \wedge c'_1 \wedge \ldots \wedge c'_h) \tag{13}$$

holds. From $\{pre(L_i)\}L_i\{post(L_i)\}$ correct we have that

$$(\mathcal{A}_\alpha \wedge c'_1 \wedge \ldots \wedge c'_h) \rightarrow post(L_i) \tag{14}$$

holds. Then from (13) and (14) it follows that

$$(\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow post(L_i) \tag{15}$$

holds. Then the claim follows from (15) and (12).

Case 2 $L_i\alpha$ is a constraint. Let $G = \leftarrow (\underline{L}_{i-1}, \underline{L}_{i+1,n})\alpha\beta_1$ be the resolvent. As $(pre(\underline{L}_{i-1}) \wedge pre(\underline{L}_{i+1,n}))\alpha\beta_1$ is true then, from the induction hypothesis applied to the subrefutation of ξ starting at G, we have that

$$(\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow (post(\underline{L}_{i-1}) \wedge post(\underline{L}_{i+1,n})). \tag{16}$$

From $\{pre(L_i)\}L_i\{post(L_i)\}$ correct it follows that either $pre(L_i) \rightarrow post(L_i)$ (if L_i is a negative equality literal) or $(pre(L_i) \wedge L_i) \rightarrow post(L_i)$ holds (if L_i is a positive equality literal). Then

$$post(L_i)\alpha\beta_1 \text{ is true.} \tag{17}$$

The claim follows from (16) and (17).

Case 3 $L_i\alpha$ is a simple positive literal. Let $C = H \leftarrow \underline{B}_m$ be the input clause. Then $\beta_1 = mgu(L_i, H)$. We distinguish the following two cases.

Case a) $m = 0$, i.e. C is a unit clause. Then the resolvent is $G = \leftarrow (\underline{L}_{i-1}, \underline{L}_{i+1,n})\alpha\beta_1$. As $(pre(\underline{L}_{i-1}) \wedge pre(\underline{L}_{i+1,n}))\alpha\beta_1$ is true then, from the induction hypothesis applied to the subrefutation of ξ starting at G, we have that

$$(\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow (post(\underline{L}_{i-1}) \wedge post(\underline{L}_{i+1,n})). \tag{18}$$

From $\{pre(L_i)\}L_i\{post(L_i)\}$ matches $\mathcal{A}P$ it follows that $(pre(L_i) \wedge H = L_i) \rightarrow post(L_i)$.

Then

$$post(L_i)\alpha\beta_1 \text{ is true.} \tag{19}$$

The claim follows from (18) and (19).

Case b) $m > 0$. Then the resolvent is $G = \leftarrow (\underline{L}_{i-1}, \underline{B}_m, \underline{L}_{i+1,n})\alpha\beta_1$.

From $\mathcal{A}P$ correct it follows that

$$\{pre(\underline{L}_{i-1}) \wedge pre(\mathcal{A}C) \wedge pre(\underline{L}_{i+1,n})\}\underline{L}_{i-1}, \underline{B}_m, \underline{L}_{i+1,n}\{post(\underline{L}_{i-1}) \wedge post(\mathcal{A}C) \wedge post(\underline{L}_{i+1,n})\}$$

is partially correct. From $\{pre(L_i)\}L_i\{post(L_i)\}$ matches $\mathcal{A}P$ it follows that

$$(pre(L_i) \wedge H = L_i) \rightarrow pre(\mathcal{A}C) \tag{20}$$

and

$$(pre(L_i) \wedge post(\mathcal{A}C) \wedge H = L_i) \rightarrow post(L_i). \tag{21}$$

From (20) it follows that $(pre(\underline{L}_{i-1}) \wedge pre(\mathcal{A}C) \wedge pre(\underline{L}_{i+1,n}))\alpha\beta_1$ is true. Then, from the induction hypothesis applied to the subrefutation of ξ starting at G, we have that

$$(\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow (post(\underline{L}_{i-1}) \wedge post(\mathcal{A}C) \wedge post(\underline{L}_{i+1,n})). \tag{22}$$

From $(pre(L_i) \wedge H = L_i)\alpha\beta_1$ true, from (21) and (22) it follows that

$$(\mathcal{A}_\alpha \wedge \mathcal{A}_{\beta_1} \wedge \ldots \wedge \mathcal{A}_{\beta_k} \wedge c) \rightarrow post(L_i) \tag{23}$$

holds. The claim follows from (22) and (23). □

Corollary 4.20 *If $\{pre(L)\}L\{post(L)\}$ is correct w.r.t. the program P then for all α and $\mathcal{A}$ s.t. $pre(L)\alpha$ is true and $\mathcal{A}$ is an answer to $L\alpha$ relative to a SLD-CNF refutation of $P \cup \{\leftarrow L\alpha\}$ the following implication holds:*

$$(\mathcal{A}_\alpha \wedge \mathcal{A}) \rightarrow post(L).$$

Example 4.21 Consider the asserted program $\mathcal{A}INC$, where the program INC is taken from [FeDe92].

$(\mathcal{A}C_1)$ `elem(X,[X|L])` $\leftarrow$
$(\mathcal{A}C_2)$ `elem(X,[H|L])` $\leftarrow$
  $\{list(L)\}$ `elem(X,L)` $\{list(L) \wedge X \in L\}$
$(\mathcal{A}C_3)$ `includ(L1,L2)` $\leftarrow$
  $\{list(L1) \wedge list(L2)\}$ $\neg$ `ninclud(L1,L2)` $\{list(L1) \wedge list(L2) \wedge L1 \subseteq L2\}$
$(\mathcal{A}C_4)$ `ninclud(L1,L2)` $\leftarrow$
  $\{list(L1)\}$ `elem(E,L1)` $\{list(L_1) \wedge E \in L1\}$,
  $\{list(L2)\}$ $\neg$ `elem(E,L2)` $\{list(L2) \wedge E \notin L2)\}$

This program defines the relation $elem(x, y)$ that holds when x is an element of the list y, the relation $includ(x, y)$ that holds when *all* elements of the list x are elements of the list y, and the relation $ninclud(x, y)$ that holds if $includ(x, y)$ does not hold, i.e. when there exists an element of the list x which does not occur in the list y.

We show that $\mathcal{A}INC$ is partially correct.

It is easy to check that INC is an acyclic program w.r.t. level mapping $|\ |$ s.t. $|elem(x, l)| = |l|$, $|includ(l1, l2)| = |l1| + |l2| + 2$ and $|ninclud(l1, l2)| = |l1| + |l2| + 1$.

We begin by showing that the negative specifications of $\mathcal{A}INC$ are correct.

Consider the specification

$$\{list(L2)\}\neg elem(E, L2)\{list(L2) \wedge E \notin L2)\}.$$

From Example 4.7 we have that

$$sp.\neg elem(E, L2).\{list(L)\} \equiv \{list(L2) \wedge \exists x_1, \ldots, x_n(L2 = [x_1, \ldots, x_n] \wedge E \neq x_1 \wedge \ldots \wedge E \neq x_n)\}.$$

Then

$$sp.\neg elem(E, L2).\{list(L2)\} \rightarrow \{list(L2) \wedge E \notin L2\}$$

holds. Hence the specification is correct.

Consider now the other negative specification

$$\{list(L1) \wedge list(L2)\}\neg ninclud(L1, L2)\{list(L1) \wedge list(L2) \wedge L1 \subseteq L2\}.$$

The implication

$$(list(L1) \wedge list(L2)) \rightarrow (\exists \underline{x}_m, \underline{y}_n(L1 = [\underline{x}_m] \wedge L2 = [\underline{y}_n]))$$

holds and $\neg ninclud([\underline{x}_m], [\underline{y}_n])$ is bounded by $n + m + 1$. It is easy to check that

$$comp(INC) \models (\neg ninclud([\underline{x}_m], [\underline{y}_n]) \leftrightarrow \bigwedge_{j \in [1,m]} (\bigvee_{i \in [1,n]} x_j = y_i)).$$

Then by Theorem 4.4 we have that

$$sp.\neg ninclud(L1, L2).\{list(L1) \wedge list(L2)\} \equiv$$
$$\{list(L1) \wedge list(L2) \wedge \exists \underline{x}_m, \underline{y}_n(L1 = [\underline{x}_m] \wedge L2 = [\underline{y}_n] \wedge \bigwedge_{j \in [1,m]}(\bigvee_{i \in [1,n]} x_j = y_i))\}.$$

Then

$$sp.\neg ninclud(L1, L2).\{list(L1) \wedge list(L2)\} \rightarrow \{list(L1) \wedge list(L2) \wedge L1 \subseteq L2\}$$

holds. Hence the specification is correct.

To conclude the proof of the partial correctness of $\mathcal{A}INC$ it is sufficient to check that the positive specification $\{list(L)\}elem(X, L)\{list(L) \wedge X \in L\}$ matches $\mathcal{A}INC$, since the other positive specification is a variant of this one. By the definition of partial correctness of a positive specification we obtain from the match of the specification with $\mathcal{A}(C_1)$ the implication

1) $(list(L) \wedge elem(X, L) = elem(X', [X'|L'])) \rightarrow (list(L) \wedge X \in L)$,

and from the match of the specification with $\mathcal{A}(C_2)$ the implications

2) $(list(L) \wedge elem(X, L) = elem(X', [H'|L'])) \rightarrow list(L')$

3) $(list(L) \wedge list(L') \wedge X' \in L' \wedge elem(X, L) = elem(X', [H'|L'])) \rightarrow (list(L) \wedge X \in L)$.

Implication 1) holds because $(X = X' \wedge L = [X'|L'])$ implies $L = [X|L']$. Implication 2) is trivial and implication 3) holds because $(X = X' \wedge L = [H'|L'] \wedge X' \in L')$ implies $X \in L$.

The partial correctness of $\mathcal{A}INC$ and Corollary 4.20 allow to conclude that:

- whenever the goal $\leftarrow elem(X, L)$ is called with L list, then at the end of every successful computation $X \in L$ holds;

- whenever the goal $\leftarrow \neg ninclud(L1, L2)$ is called with $L1$ and $L2$ lists, then at the end of every successful computation $L1 \subseteq L2$ holds;

- whenever the goal $\leftarrow \neg elem(X, L)$ is called with L list, then at the end of every successful computation $X \notin L$ holds.

Finally it is easy to check that the specification

$$\{list(L1) \wedge list(L2)\}\ includ(L1, L2)\ \{list(L1) \wedge list(L2) \wedge L1 \subseteq L2\}$$

is partially correct w.r.t. INC. In fact $\mathcal{A}INC$ is partially correct and the specification matches the asserted program because

- $(list(L1) \wedge list(L2) \wedge includ(L1, L2) = includ(L1', L2')) \rightarrow (list(L1') \wedge list(L2'))$

and

- $(list(L1') \wedge list(L2') \wedge L1' \subseteq L2' \wedge includ(L1, L2) = includ(L1', L2')) \rightarrow (list(L1) \wedge list(L2) \wedge L1 \subseteq L2)$,

are both valid implications.

4.3 Extension to General Programs

It has been shown in Theorem 4.4 how for acyclic (resp. acceptable) programs P and negative literals L the strongest postcondition can be computed using the property of bounded goals of been equality definable w.r.t. $comp(P)$. It is easy to check that the condition "P acyclic" (resp. "P acceptable") can be weakened by taking the condition "P_L acyclic" (resp. "P_L acceptable"), where P_L is the set of clauses of P in whose head the relation p of L or the relations on which p *depends on* occurs. In fact the condition "P acyclic" (resp. "P acceptable") is used in the proof of Theorem 4.4 to guarantee that the SLD-CNF (resp. LD-CNF) tree of $P \cup \{\leftarrow L\}$ is finite, when L is bounded. But if P_L is acyclic then the SLD-CNF (resp. LD-CNF) tree of $P \cup \{\leftarrow L\}$ is finite, since the relations occurring in the tree are only from P_L. As a consequence the condition "P acyclic" (resp. "P acceptable") in the Definition 4.11 of correctness of an asserted program can be weakened by taking the condition "Neg^*_P acyclic" (resp. "Neg^*_P acceptable") .

Example 4.22 Let $con_e(x_0, y_0)$ be the predicate which is true when x_0 and y_0 are nodes of the graph e that are connected without cycles, i.e.

$$\exists \underline{x}_n (\forall i, j \in [0, n]((x_i \neq y_0) \wedge (i \neq j \rightarrow x_i \neq x_j)) \wedge (\forall i \in [0, n-1][x_i, x_{i+1}] \in e) \wedge [x_n, y_0] \in e).$$

Consider the following asserted version $\mathcal{A}TRANS$ of the program $TRANS$ that computes the transitive closure of a graph.

```
trans(X, Y, E, V) ←  {list(E)} elem([X, Y], E) {[X,Y] ∈ E}
trans(X, Z, E, V) ←
    {list(E)} elem([X, Y], E) {[X,Y] ∈ E},
    {list(V)} ¬ elem(Y, V) {Y ∉ V},
    {list(E)} trans(Y, Z, E, [Y | V]) {con_E(Y,Z)}
elem(X,[X|L]) ←
elem(X,[H|L]) ←
    {list(L)} elem(X,L) {list(L) ∧ X ∈ L}
```

In [ApPe91] it has been shown that $TRANS$ is not acyclic. However Neg^*_{TRANS} is acyclic, since it reduces to the set of clauses that define the relation $elem/2$ (see Example 4.7). Consider the specification

$$\{p\}\ trans(X, Y, e, [\,])\ \{q\},$$

where

$$p = (list(e) \wedge \forall x(x \in e \rightarrow x \in \mathcal{N}))$$

and

$$q = (p \wedge con_e(X, Y)),$$

with $\mathcal{N}$ finite set of nodes. We show that $\mathcal{A}TRANS$ is correct and that the specification matches $\mathcal{A}TRANS$.

From Example 4.7 to prove the partial correctness of the negative specification $\{list(V)\}\ \neg elem(Y, V)\ \{Y \notin V\}$ reduces to the implication

$$(list(V) \wedge \exists \underline{x}_n(elem(Y, V) = elem(Y, [\underline{x}_n]) \wedge \forall i \in [1, n] Y \neq x_i)) \rightarrow (Y \notin V)$$

which clearly holds.

The proof that $\{list(E)\}\ elem([X, Y], E)\ \{[X, Y] \in E\}$ matches $\mathcal{A}TRANS$ reduces to the following implications.

- $(list(E) \wedge elem([X, Y], E) = elem(X', [X'|L'])) \rightarrow [X, Y], \in E$,
  from the match of the specification with the third clause of $\mathcal{A}TRANS$;
- $(list(E) \wedge elem([X, Y], E) = elem(X', [H'|L'])) \rightarrow list(L')$ and
- $(list(E) \wedge elem([X, Y], E) = elem(X', [H'|L']) \wedge list(L') \wedge X' \in L') \rightarrow [X, Y] \in E$,
  from the match of the specification with the fourth clause of $\mathcal{A}TRANS$.

It is immediate to check that these are all valid implications.

The proof that $\{list(L)\}\ elem(X, L)\ \{list(L) \wedge X \in L\}$ matches $\mathcal{A}TRANS$ is analogous to the previous one.

It remains to show that $\{list(E)\}\ trans(Y, Z, E, [Y|V])\ \{con_E(Y, Z)\}$ matches $\mathcal{A}TRANS$. We obtain the following two implications, which trivially hold.

- $(list(E) \wedge trans(Y, Z, E, [Y|V]) = trans(Y', Z', E', [Y'|V'])) \rightarrow list(E')$,
- $(list(E) \wedge trans(Y, Z, E, [Y|V]) = trans(Y', Z', E', [Y'|V']) \wedge con_{E'}(Y', Z')) \rightarrow con_E(Y, Z)$.

Acknowledgements

I would like to thank Krzysztof Apt for his suggestion to study properties of acyclic and acceptable programs w.r.t. constructive negation and for his comments on the presentation. I would also like to thank Frank Teusink for his comments, Livio Colussi and Włodek Drabent for discussions on the subject of partial correctness of logic programs.

References

[AnGa91] S. Antoy, J. Gannon. Axiomatic Verification of Logic Programs. *TR 90-26.2*, Portland State University, 1991.

[ApDo92] K. R. Apt., K. Doets. A new definition of SLDNF-resolution. *TR* Department of Mathematics and Computer Science, University of Amsterdam, The Netherlands, 1992. To appear.

[ApBe91] K. R. Apt, M. Bezem. Acyclic Programs. *New Generation Computing*, Vol. 9, 335–363, 1991.

[ApPe90] K. R. Apt, D. Pedreschi. Studies in pure Prolog: Termination. *Symposium on Computational Logic*, J. W. Lloyd, editor, 150–176, Berlin, Springer-Verlag, 1990.

[ApPe91] K. R. Apt, D. Pedreschi. Proving Termination of General Prolog Programs. *Report CS-R911*, Centre for Mathematics and Computer Science,1991.

[Bezem90] M. Bezem. Characterizing Termination of Logic Programs with Level Mappings. *Proceedings of the 1990 North American Conference on Logic Programming* , 1990.

[BoCo89] A. Bossi, N. Cocco. Verifying Correctness of Logic Programs. *Proceedings of Tapsoft '89*, LNCS, pp. 96-110, 1989.

[Cav89] L. Cavedon. Continuity, Consistency and Completeness Properties of Logic Programs. *Proceedings of the 6th International Conference on Logic Programming*, The MIT Press, pp.571-584, 1989.

[Chan88] D.Chan. Constructive Negation Based on the Completed Database. *Proceedings of the 9th International Conference and Symposium on Logic Programming*, 1988.

[CoMa91] L. Colussi, E. Marchiori. Proving Correctness of Logic Programs Using Axiomatic Semantics. *Proceedings of the 8th International Conference on Logic Programming*, The MIT Press, pp. 629-644, 1991.

[CoMa92] L. Colussi, E. Marchiori. A Predicate Transformer for Unification. *Proceedings of the '92 Joint International Conference and Symposium on Logic Programming*, The MIT Press, 1992.

[Der89] P. Deransart. Proofs of Declarative Properties of Logic Programs. *Proceedings of Tapsoft '89*, LNCS, pp. 207-226, 1989.

[Der87] N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3, pp. 69-116, 1987.

[Dr91] W. Drabent. On Axiomatic Semantics for Logic Programs. *Workshop on Verification of Logic Program Properties: Termination*, University of Padua, Department of Pure and Applied Mathematics, Padua, October 1991.

[FeDe92] G. Ferrand, P. Deransart. Proof Method of Partial Correctness and Partial Completeness of Normal Logic Programs. *Proceedings of the 92 Joint International Conference and Symposium on Logic Programming*, The MIT Press,1992.

[DrMa87] W. Drabent, J. Maluszynski. Inductive Assertion Method for Logic Programs. *Proceedings of Tapsoft '87*, LNCS 250, 1987.

[HaMcDe87] S. Hanks, D. McDermott. Nonmonotonic Logic and Temporal Reasoning. *Artificial Intelligence*, 33, pp. 379-412, 1987.

[Hoare69] C.A.R. Hoare. An Axiomatic Basis for Computer Programming. *C.A.C.M.*, vol.12, pp. 576-580, 1969.

[Prz89] T. C. Przymusinski. On Constructive Negation in Logic Programming. *Proceedings of the North American Logic Programming Conference*, Cleveland, Ohio, October 1989.

[SS86] , L. Sterling, E. Shapiro. The Art of Prolog. *MIT Press*, 1986.