Reasoning about Prolog programs: from modes through types to assertions

K.R. Apt, E. Marchiori

Computer Science/Department of Software Technology

Report CS-R9358 August 1993

CWI is the National Research Institute for Mathematics and Computer Science. CWI is part of the Stichting Mathematisch Centrum (SMC), the Dutch foundation for promotion of mathematics and computer science and their applications.
SMC is sponsored by the Netherlands Organization for Scientific Research (NWO). CWI is a member of ERCIM, the European Research Consortium for Informatics and Mathematics.

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

Reasoning about Prolog Programs: from Modes through Types to Assertions

Krzysztof R. Apt
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
and
Faculty of Mathematics and Computer Science,
University of Amsterdam, Plantage Muidergracht 24
1018 TV Amsterdam, The Netherlands

Elena Marchiori
CWI
P.O. Box 494079, 1090 GB Amsterdam, The Netherlands

Abstract

We analyze here relative strength of a number of methods for program analysis in logic programming, and study the relationship between them. We show that these methods form the following hierarchy: mode analysis ⇒ type analysis ⇒ monotonic properties ⇒ run time properties.

1991 Mathematics Subject Classification: 68Q40, 68T15, 68N17
1991 CR Categories: F.3.1., F.4.1, H.3.3, I.2.2
Keywords and Phrases: Prolog programs, program verification
Note: This research was partly supported by the ESPRIT Basic Research Action 6810 (Compulog 2)

1 Introduction

1.1 Motivation

Over the past 8 years a number of proposals were made in the literature for the analysis and verification of logic programs, based on the concepts of modes, types and assertions, both monotonic ones and non-monotononic ones (like $var(X)$). The aim of this paper is to show that these methods can be arranged in a hierarchy in which increasingly stronger program properties can be established and in which each method is a generalization of the preceding ones.

More specifically, we deal here with the following notions: well-moded programs, essentially due to Dembinski and Maluszynski [DM85], well-typed programs, due to Bronsard, Lakshman and Reddy [BLR92], the assertional method of Bossi and Cocco [BC89], the assertional method of Drabent and Małuszyński [DM88] and the assertional method of Colussi and Marchiori [CM91].

We believe that the systematic presentation of these methods of program analysis is useful for a number of reasons. First it clarifies the relationship between them. Next, it allows us to justify them by means of simpler correctness proofs than the original ones. Finally, it allows us to better understand which program properties can be established by means of which method.

1.2 Preliminaries

We consider logic programs executed by means of the *LD-resolution*, which consists of the SLD-resolution combined with the leftmost selection rule. An SLD-derivation in which the leftmost selection rule is used

Report CS-R9358
ISSN 0169-118X
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

is called an *LD-derivation*.

We work here with *queries*, that is sequences of atoms, instead of *goals*, that is constructs of the form $\leftarrow Q$, where Q is a query. Apart from this we use the standard notation of Lloyd [Llo87] and Apt [Apt90]. In particular, given a syntactic construct E (so for example, a term, an atom or a set of equations) we denote by $vars(E)$ the set of the variables appearing in E. Given a substitution $\theta = \{x_1/t_1, ..., x_n/t_n\}$ we denote by $dom(\theta)$ the set of variables $\{x_1, \ldots, x_n\}$ and by $range(\theta)$ the set of variables appearing in $\{t_1, \ldots, t_n\}$. Finally, we define $vars(\theta) = dom(\theta) \cup range(\theta)$.

2 Well-moded Programs

We start by introducing modes. They were first considered in Mellish [Mel81], and more extensively studied in Reddy [Red84], [Red86] and Dembinski and Maluszynski [DM85].

Definition 2.1 Consider an n-ary relation symbol p. By a *mode* for p we mean a function m_p from $\{1, \ldots, n\}$ to the set $\{+, -\}$. If $m_p(i) = '+'$, we call i an *input position* of p and if $m_p(i) = '-'$, we call i an *output position* of p (both w.r.t. m_p).

We write m_p in a more suggestive form $p(m_p(1), \ldots, m_p(n))$. By a *moding* we mean a collection of modes, each for a different relation symbol. □

Modes indicate how the arguments of a relation should be used. The definition of moding assumes one mode per relation in a program. Multiple modes may be obtained by simply renaming the relations. In the remainder of this section we adopt the following

Assumption 2.2 *Every considered relation* has a fixed mode associated with it.

This will allow us to talk about input positions and output positions of an atom.

We now introduce the notion of a well-moded program. The concept is due to Dembinski and Maluszynski [DM85]; we use here an elegant formulation due to Rosenblueth [Ros91] (which is equivalent to that of Drabent [Dra87] where well-moded programs are called simple). The definition of a well-moded program constrains the "flow of data" through the clauses of the programs. To simplify the notation, when writing an atom as $p(\mathbf{u}, \mathbf{v})$, we now assume that $\mathbf{u}$ is a sequence of terms filling in the input positions of p and that $\mathbf{v}$ is a sequence of terms filling in the output positions of p.

Definition 2.3

- A query $p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$ is called *well-moded* if for $i \in [1, n]$

$$vars(\mathbf{s_i}) \subseteq \bigcup_{j=1}^{i-1} vars(\mathbf{t_j}).$$

- A clause

$$p_0(\mathbf{t_0}, \mathbf{s_{n+1}}) \leftarrow p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$$

  is called *well-moded* if for $i \in [1, n+1]$

$$vars(\mathbf{s_i}) \subseteq \bigcup_{j=0}^{i-1} vars(\mathbf{t_j}).$$

- A program is called *well-moded* if every clause of it is. □

Thus, a query is well-moded if

- every variable occurring in an input position of an atom ($i \in [1, n]$) occurs in an output position of an earlier ($j \in [1, i-1]$) atom.

And a clause is well-moded if

- $(i \in [1,n])$ every variable occurring in an input position of a body atom occurs either in an input position of the head $(j = 0)$, or in an output position of an earlier $(j \in [1, i-1])$ body atom,
- $(i = n+1)$ every variable occurring in an output position of the head occurs in an input position of the head $(j = 0)$, or in an output position of a body atom $(j \in [1,n])$.

Note that a query with only one atom is well-moded iff this atom is ground in its input positions. The following notion is due to Dembinski and Maluszynski [DM85].

Definition 2.4 We call an LD-derivation *data driven* if all atoms selected in it are ground in their input positions. □

The following lemma shows the "persistence" of the notion of well-modedness.

Lemma 2.5 *An LD-resolvent of a well-moded query and a well-moded clause that is variable disjoint with it, is well-moded.*

Proof. An LD-resolvent of a query and a clause is obtained by means of the following three operations:

- instantiation of a query,
- instantiation of a clause,
- replacement of the first atom, say H, of a query by the body of a clause whose head is H.

So we only need to prove the following two claims.

Claim 1 *An instance of a well-moded query (resp. clause) is well-moded.*

Proof. It suffices to note that for any sequences of terms $\mathbf{s}, \mathbf{t_1}, \ldots, \mathbf{t_n}$ and a substitution σ, $vars(\mathbf{s}) \subseteq \bigcup_{j=1}^{n} vars(\mathbf{t_j})$ implies $vars(\mathbf{s}\sigma) \subseteq \bigcup_{j=1}^{n} vars(\mathbf{t_j}\sigma)$. □

Claim 2 *Suppose $H, \mathbf{A}$ is a well-moded query and $H \leftarrow \mathbf{B}$ is a well-moded clause. Then $\mathbf{B}, \mathbf{A}$ is a well-moded query.*

Proof. Let $H = p(\mathbf{s}, \mathbf{t})$ and $\mathbf{B} = p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$. We have $vars(\mathbf{s}) = \emptyset$ since H is the first atom of a well-moded query. Thus $\mathbf{B}$ is well-moded. Moreover, $vars(\mathbf{t}) \subseteq \bigcup_{j=1}^{n} vars(\mathbf{t_j})$, since $H \leftarrow \mathbf{B}$ is a well-moded clause and $vars(\mathbf{s}) = \emptyset$. These two observations imply the claim. □

The definition of a well-moded program is designed in such a way that the following theorem, also due to Dembinski and Maluszynski [DM85], holds.

Theorem 2.6 *Let P and Q be well-moded. Then all LD-derivations of Q in P are data driven.*

Proof. Note that the first atom of a well-moded query is ground in its input positions and a variant of a well-moded clause is well-moded. The conclusion now follows by Lemma 2.5. □

The following is a well-known conclusion of this theorem.

Corollary 2.7 *Let P and Q be well-moded. Then for every computed answer substitution σ, $Q\sigma$ is ground.*

Proof. Let $\mathbf{x}$ stand for the sequence of all variables that appear in Q. Let p be a new relation of arity equal to the length of $\mathbf{x}$ and with all positions moded as input. Then $Q, p(\mathbf{x})$ is a well-moded query.

Now, σ is a computed answer substitution for Q in P iff $p(\mathbf{x})\sigma$ is a selected atom in an LD-derivation of $Q, p(\mathbf{x})$ in P. The conclusion now follows by Theorem 2.6. □

Let us see now how these results can be applied to specific programs.

Example 2.8 Consider the program `quicksort`:

```
qs([X | Xs], Ys) ←
   partition(X, Xs, Littles, Bigs),
   qs(Littles, Ls),
   qs(Bigs, Bs),
   app(Ls, [X | Bs], Ys).
qs([], []).

partition(X, [Y | Xs], [Y | Ls], Bs) ← X > Y, partition(X, Xs, Ls, Bs).
partition(X, [Y | Xs], Ls, [Y | Bs]) ← X ≤ Y, partition(X, Xs, Ls, Bs).
partition(X, [], [], []).

app([X|Xs], Ys, [X|Zs]) ← app(Xs, Ys, Zs).
app([], Ys, Ys).
```

Conforming to Prolog behaviour, we assume that the evaluation of the tests $u > v$ and $u \leq v$ ends in an error if u or v are not ground arithmetic expressions (in short, gae's). We mode it as follows: `qs(+,-)`, `partition(+,+,-,-)`, `app(+,+,-)`. It is easy to check that `quicksort` is then well-moded. Assume now that s is a ground term. By Theorem 2.6 all LD-derivations of `qs(s, t)` in `quicksort` are data driven and by Corollary 2.7 we conclude that all the computed answer substitutions σ are such that $t\sigma$ is ground. □

In conclusion, mode analysis is sufficient to derive information on groundness of atom arguments, before or after their selection.

3 Well-typed Programs

3.1 Types and Type Judgements

To deal with run time errors we introduce the notion of a type. We adopt the following general definition.

Definition 3.1 A *type* is a decidable set of terms closed under substitution. □

Certain types will be of special interest:
List — the set of lists,
Gae — the set of of gae's,
ListGae — the set of lists of gae's.
Ground — the set of ground terms.

Of course, the use of the type *List* assumes the existence of the empty list `[]` and the list constructor `[.|.]` in the language, the use of the type *Gae* assumes the existence of the numeral `0` and the successor function `s(.)` and the use of the type *ListGae* assumes the existence of what the use of the types *List* and *Gae* implies.

Throughout the paper we fix a specific set of types, denoted by *Types*, which includes the above ones. We call a construct of the form $s : S$, where s is a term and S is a type, a *typed term*. Given a sequence $\mathbf{s} : \mathbf{S} = s_1 : S_1, \ldots, s_n : S_n$ of typed terms, we write $\mathbf{s} \in \mathbf{S}$ if for $i \in [1, n]$ we have $s_i \in S_i$, and define $vars(\mathbf{s} : \mathbf{S}) = vars(\mathbf{s})$. Further, we abbreviate the sequence $s_1\theta, ..., s_n\theta$ to $\mathbf{s}\theta$. We say that $\mathbf{s} : \mathbf{S}$ is *realizable* if $\mathbf{s}\eta \in \mathbf{S}$ for some η.

Definition 3.2

- By a *type judgement* we mean a statement of the form

$$\mathbf{s} : \mathbf{S} \Rightarrow \mathbf{t} : \mathbf{T}. \tag{1}$$

- We say that a type judgement (1) *is true*, and write

$$\models \mathbf{s} : \mathbf{S} \Rightarrow \mathbf{t} : \mathbf{T},$$

if for all substitutions θ, $\mathbf{s}\theta \in \mathbf{S}$ implies $\mathbf{t}\theta \in \mathbf{T}$.

□

For example, the type judgement $s(s(x))$: *Gae*, l : *ListGae* $\Rightarrow$ $[x \mid l]$: *ListGae* is true. How to prove that a type judgement is true is an interesting problem but irrelevant for our considerations. In all considered cases it will be clear how to show that a type judgement is true.

The following simple properties of type judgements hold.

Lemma 3.3 (Type Judgement) *Let $\phi, \phi_1, \phi_2, \phi_2', \phi_3$ and ψ be sequences of typed terms.*

(i) Suppose that $s \in S$ and

$$\models s : S, \phi \Rightarrow \psi.$$

Then

$$\models \phi \Rightarrow \psi.$$

(ii) Suppose that

$$\models \phi_2 \Rightarrow \phi_2'$$

and

$$\models \phi_1, \phi_2', \phi_3 \Rightarrow \psi.$$

Then

$$\models \phi_1, \phi_2, \phi_3 \Rightarrow \psi.$$

(iii) Suppose that

$$\models \mathbf{s} : \mathbf{S}, \mathbf{t} : \mathbf{T} \Rightarrow \mathbf{u} : \mathbf{U},$$

$$\mathbf{t} : \mathbf{T} \textit{ is realizable,}$$

and

$$vars(\mathbf{t}) \cap vars(\mathbf{s}, \mathbf{u}) = \emptyset.$$

Then

$$\models \mathbf{s} : \mathbf{S} \Rightarrow \mathbf{u} : \mathbf{U}.$$

Proof.
(i) By the assumption that all types are closed under substitution.
(ii) Immediate.
(iii) Take θ such that $\mathbf{s}\theta \in \mathbf{S}$ and let η be such that $\mathbf{t}\eta \in \mathbf{T}$. Define $\theta' = \theta_{|vars(\mathbf{s},\mathbf{u})}$ and $\eta' = \eta_{|vars(\mathbf{t})}$. Then $\sigma = \theta' \cup \eta'$ is well-defined, $\mathbf{s}\sigma \in \mathbf{S}$ and $\mathbf{t}\sigma \in \mathbf{T}$. So $\mathbf{u}\sigma \in \mathbf{U}$, i.e. $\mathbf{u}\theta \in \mathbf{U}$. □

3.2 Well-typed Queries and Programs

The next step is to define types for relations.

Definition 3.4 Consider an n-ary relation symbol p. By a *type* for p we mean a function t_p from $[1, n]$ to the set *Types*. If $t_p(i) = i$, we call *i the type associated with the position i of p.* □

In the remainder of this section we consider a combination of modes and types and adopt the following

Assumption *Every considered relation* has a fixed mode and a fixed type associated with it. □

This assumption will allow us to talk about types of input positions and of output positions of an atom. An n-ary relation p with a mode m_p and type t_p will be denoted by

$$p(m_p(1) : t_p(1), \ldots, m_p(n) : t_p(n)).$$

For example, app(+ : *List*, + : *List*, − : *List*) denotes a ternary relation app with the first two positions moded as input and typed as *List*, and the third position moded as output and typed as *List*.

To simplify the notation, when writing an atom as $p(\mathbf{u} : \mathbf{S}, \mathbf{v} : \mathbf{T})$ we now assume that $\mathbf{u} : \mathbf{S}$ is a sequence of typed terms filling in the input positions of p and $\mathbf{v} : \mathbf{T}$ is a sequence of typed terms filling in the output positions of p. We call a construct of the form $p(\mathbf{u} : \mathbf{S}, \mathbf{v} : \mathbf{T})$ a *typed atom*. We say that a typed atom $p(s_1 : S_1, \ldots, s_n : S_n)$ is *correctly typed in position i* if $s_i \in S_i$.

The following notion is due to Bronsard, Lakshman and Reddy [BLR92].

Definition 3.5

- A query $p_1(\mathbf{i_1} : \mathbf{I_1}, \mathbf{o_1} : \mathbf{O_1}), \ldots, p_n(\mathbf{i_n} : \mathbf{I_n}, \mathbf{o_n} : \mathbf{O_n})$ is called *well-typed* if for $j \in [1, n]$

$$\models \mathbf{o_1} : \mathbf{O_1}, \ldots, \mathbf{o_{j-1}} : \mathbf{O_{j-1}} \Rightarrow \mathbf{i_j} : \mathbf{I_j}.$$

- A clause $p_0(\mathbf{o_0} : \mathbf{O_0}, \mathbf{i_{n+1}} : \mathbf{I_{n+1}}) \leftarrow p_1(\mathbf{i_1} : \mathbf{I_1}, \mathbf{o_1} : \mathbf{O_1}), \ldots, p_n(\mathbf{i_n} : \mathbf{I_n}, \mathbf{o_n} : \mathbf{O_n})$ is called *well-typed* if for $j \in [1, n+1]$

$$\models \mathbf{o_0} : \mathbf{O_0}, \ldots, \mathbf{o_{j-1}} : \mathbf{O_{j-1}} \Rightarrow \mathbf{i_j} : \mathbf{I_j}.$$

- A program is called *well-typed* if every clause of it is. □

Thus, a query is well-typed if

- the types of the terms filling in the *input* positions of an atom can be deduced from the types of the terms filling in the *output* positions of the previous atoms.

And a clause is well-typed if

- ($j \in [1, n]$) the types of the terms filling the *input* positions of a body atom can be deduced from the types of the terms filling in the *input* positions of the head and the *output* positions of the previous body atoms,
- ($j = n + 1$) the types of the terms filling in the *output* positions of the head can be deduced from the types of the terms filling in the *input* positions of the head and the types of the terms filling in the *output* positions of the body atoms.

Note that a query with only one atom is well-typed iff this atom is correctly typed in its input positions. The following observation clarifies the relation between well-moded and well-typed programs and queries.

Theorem 3.6 *The notion of well-moded program (resp. query) is a special case of the notiton of well-typed program (resp. query).*

Proof. Take *Ground* as the only type. Then the notions of a well-moded program (resp. query) and a well-typed program (resp. query) coincide. □

The following lemma stated in Bronsard, Lakshman and Reddy [BLR92] shows persistence of the notion of being well-typed.

Lemma 3.7 *An LD-resolvent of a well-typed query and a well-typed clause that is variable disjoint with it, is well-typed.*

Proof. We reason as in the proof of Lemma 2.5. So we need to prove the following two claims.

Claim 1 *An instance of a well-typed query (resp. clause) is well-typed.*

Proof. Immediate by definition. □

Claim 2 *Suppose $H, \mathbf{A}$ is a well-typed query and $H \leftarrow \mathbf{B}$ is a well-typed clause. Then $\mathbf{B}, \mathbf{A}$ is a well-typed query.*

Proof. Let $H = p(\mathbf{s} : \mathbf{S}, \mathbf{t} : \mathbf{T})$ and $\mathbf{B} = p_1(\mathbf{i_1} : \mathbf{I_1}, \mathbf{o_1} : \mathbf{O_1}), \ldots, p_m(\mathbf{i_m} : \mathbf{I_m}, \mathbf{o_m} : \mathbf{O_m})$. H is the first atom of a well-typed query, so it is correctly typed in its input positions, i.e.

$$\mathbf{s} \in \mathbf{S}. \tag{2}$$

$H \leftarrow \mathbf{B}$ is well-typed, so

$$\models \mathbf{s} : \mathbf{S}, \mathbf{o_1} : \mathbf{O_1}, \ldots, \mathbf{o_m} : \mathbf{O_m} \Rightarrow \mathbf{t} : \mathbf{T},$$

and for $j \in [1, m]$

$$\models \mathbf{s} : \mathbf{S}, \mathbf{o_1} : \mathbf{O_1}, \ldots, \mathbf{o_{j-1}} : \mathbf{O_{j-1}} \Rightarrow \mathbf{i_j} : \mathbf{I_j}.$$

By the Type Judgement Lemma 3.3(i) we get by virtue of (2)

$$\models \mathbf{o_1} : \mathbf{O_1}, \ldots, \mathbf{o_m} : \mathbf{O_m} \Rightarrow \mathbf{t} : \mathbf{T}, \tag{3}$$

and for $j \in [1, m]$

$$\models \mathbf{o_1} : \mathbf{O_1}, \ldots, \mathbf{o_{j-1}} : \mathbf{O_{j-1}} \Rightarrow \mathbf{i_j} : \mathbf{I_j}. \tag{4}$$

Now, let $\mathbf{A} = p_{m+1}(\mathbf{i_{m+1}} : \mathbf{I_{m+1}}, \mathbf{o_{m+1}} : \mathbf{O_{m+1}}), \ldots, p_n(\mathbf{i_n} : \mathbf{I_n}, \mathbf{o_n} : \mathbf{O_n})$. $H, \mathbf{A}$ is well-typed, so for $j \in [m+1, n]$

$$\models \mathbf{t} : \mathbf{T}, \mathbf{o_{m+1}} : \mathbf{O_{m+1}}, \ldots, \mathbf{o_{j-1}} : \mathbf{O_{j-1}} \Rightarrow \mathbf{i_j} : \mathbf{I_j},$$

and thus by (3) and the Type Judgement Lemma 3.3(ii) for $j \in [m+1, n]$

$$\models \mathbf{o_1} : \mathbf{O_1}, \ldots, \mathbf{o_{j-1}} : \mathbf{O_{j-1}} \Rightarrow \mathbf{i_j} : \mathbf{I_j}.$$

This and (4) imply the claim. □

□

This brings us to the following desired conclusions.

Theorem 3.8 *Let P and Q be well-typed and let ξ be an LD-derivation of Q in P. All atoms selected in ξ are correctly typed in their input positions.*

Proof. Note that the first atom of a well-typed query is correctly typed in its input positions and that a variant of a well-typed clause is well-typed. The conclusion now follows by Lemma 3.7. □

Corollary 3.9 *Let P and Q be well-typed. Then for every computed answer substitution σ, $Q\sigma$ is well-typed in its output positions.*

Proof. Let $\mathbf{o} : \mathbf{O}$ stand for the sequence of typed terms filling in the output positions of the atoms of Q. Let p be a new relation of arity equal to the length of $\mathbf{o} : \mathbf{O}$ and with all the positions moded as input and typed as $\mathbf{O}$. Then $Q, p(\mathbf{o} : \mathbf{O})$ is a well-typed query. Now, σ is a computed answer substitution for Q in P iff $p(\mathbf{o})\sigma$ is a selected atom in an LD-derivation of $Q, p(\mathbf{o})$ in P. The conclusion now follows by Theorem 3.8. □

Let us see now how these results can be applied to specific programs.

Example 3.10 Reconsider the program `quicksort`. We type it as follows:

```
qs(+:ListGae, -:ListGae),
partition(+:Gae, +:ListGae, -:ListGae, -:ListGae),
app(+:ListGae, +:ListGae, -:ListGae),
>(+:Gae, +:Gae), ≤(+:Gae, +:Gae).
```

It is easy to check that `quicksort` is then well-typed. Assume now that s is a list of natural numbers. By Theorem 3.8 we conclude that all atoms selected in the LD-derivations of `qs(s,t)` in `quicksort` are correctly typed in their input positions. In particular, when these atoms are of the form $u > v$ or $u \leq v$, both u and v are gae's. Thus the LD-derivations of `qs(s,t)` do not end in an error. Moreover, by Corollary 3.9 we conclude that all computed answer substitutions σ are such that $t\sigma$ is a list of gae's. □

Thus, type analysis is sufficient to derive information about the types of atom arguments, before or after their selection. This is sufficient to prove absence of run time errors.

4 Asserted programs

To deal with correctness of programs we need to use assertions. Assertions are formed in an extension of a first-order language. A pair of assertions (called pre- and post-condition) is associated with every atom of the program, describing, respectively, the properties of the arguments of the atom before and after its execution. We shall see that the form of the assertions one allows can affect the simplicity of the method. In particular, the use of monotonic assertions results in a simpler method. Intuitively an assertion is monotonic when it remains true under instantiation. For instance, $ground(x)$ is a monotonic assertion and $var(x)$ is a non-monotonic assertions, since any instance of a ground term is ground, and in general instances of a variable are not variables. Formally, an assertion ϕ is *monotonic* if for every substitution σ

$$\models \phi \Rightarrow \phi\sigma. \tag{5}$$

To render the exposition uniform, the formalisms discussed here will sometimes slightly differ from those of the original works.

4.1 Well-asserted programs

To deal with monotonic run time properties we introduce the notion of well-asserted program. This noption was introduced by Bossi and Cocco in [BC89]. Let P be a program. The *assertion language for* P contains variables x_1^p, ..., x_n^p, for every n-ary relation p of P. These variables are called below p-variables.

Definition 4.1 (specification) A *specification for an n-ary relation* p is a pair $(pre^p, post^p)$ of monotonic assertions, s.t. $vars(pre^p, post^p) \subseteq \{x_1^p, \ldots, x_n^p\}$. □

An *asserted program* $\mathcal{AP}$ is obtained by specifying a specification for every relation of P. In the remainder of this section we adopt the following

Assumption 4.2 *Every specific relation* has a fixed specification associated with it.

Definition 4.3 Let $A = p(t_1, \ldots, t_n)$ and $\alpha = \{x_i^p/t_i \mid i \in [1,n]\}$. We denote by $pre(A)$ the assertion $pre^p\alpha$ and by $post(A)$ the assertion $post^p\alpha$. We use $pre(A_1, \ldots, A_k)$ as a shorthand for $pre(A_1) \wedge \ldots \wedge pre(A_k)$, and similarly with *post*.

- We say that A *satisfies its precondition* if $\models pre(A)$,
- We say that A *satisfies its postcondition* if $\models post(A)$.

Definition 4.4 (well-asserted program)

- A query $B_1, \ldots, B_n$ is called *well-asserted* if for $j \in [1,n]$

$$\models post(B_1, \ldots, B_{j-1}) \Rightarrow pre(B_j).$$

- A clause $H \leftarrow B_1, \ldots, B_n$ is called *well-asserted* if for $j \in [1, n+1]$

$$\models pre(H) \wedge post(B_1, \ldots, B_{j-1}) \Rightarrow pre(B_j),$$

  where $pre(B_{n+1}) \stackrel{\text{def}}{=} post(H)$.

- An asserted program $\mathcal{AP}$ is called *well-asserted* if every clause of it is. □

When ambiguity does not arise we say that P is well-asserted instead of $\mathcal{AP}$ well-asserted. The following observation clarifies the relation between well-asserted and well-typed programs and queries.

Theorem 4.5 *The notion of a well-typed program (resp. query) is a special case of the notion of a well-asserted program (resp. query).*

Proof. It suffices to view a typed atom $p(\mathbf{x} : \mathbf{S}, \mathbf{y} : \mathbf{T})$ as a specification for the relation $p(\mathbf{x}, \mathbf{y})$ consisting of $pre^p = \mathbf{x} \in \mathbf{S}$ and $post^p = \mathbf{y} \in \mathbf{T}$. Then a program P is well-typed iff the corresponding asserted program is well-asserted. □

The following lemma shows persistence of the notion of being well-asserted.

Lemma 4.6 *An LD-resolvent of a well-asserted query and a well-asserted clause that is variable disjoint with it, is well-asserted.*

Proof. We reason as in the proof of Lemma 2.5. We need to prove the following two claims.

Claim 1 *An instance of a well-asserted query (resp. clause) is well-asserted.*

Proof. Immediate by the assumption that the assertions are monotonic. □

Claim 2 *Suppose $H, \mathbf{A}$ is a well-asserted query and $H \leftarrow \mathbf{B}$ is a well-asserted clause. Then $\mathbf{B}, \mathbf{A}$ is a well-asserted query.*

Proof. Let $H = p(\mathbf{s})$ and $\mathbf{B} = p_1(\mathbf{s_1}), \ldots, p_m(\mathbf{s_m})$. H is the first atom of a well-asserted query, so it satisfies its precondition, i.e.

$$\models pre(p(\mathbf{s})). \tag{6}$$

$H \leftarrow \mathbf{B}$ is well-asserted, so

$$\models pre(p(\mathbf{s})) \wedge post(p_1(\mathbf{s_1}), \ldots, p_m(\mathbf{s_m})) \Rightarrow post(p(\mathbf{s})),$$

and for $j \in [1, m]$

$$\models pre(p(\mathbf{s})) \wedge post(p_1(\mathbf{s_1}), \ldots, p_{j-1}(\mathbf{s_{j-1}})) \Rightarrow pre(p(\mathbf{s_j})).$$

By (6) we obtain

$$\models post(p_1(\mathbf{s_1}), \ldots, p_m(\mathbf{s_m})) \Rightarrow post(p(\mathbf{s})), \tag{7}$$

and for $j \in [1, m]$

$$\models post(p_1(\mathbf{s_1}), \ldots, p_{j-1}(\mathbf{s_{j-1}})) \Rightarrow pre(p(\mathbf{s_j})). \tag{8}$$

Now, let $\mathbf{A} = p_{m+1}(\mathbf{s_{m+1}}), \ldots, p_n(\mathbf{s_n})$. $H, \mathbf{A}$ is a well-asserted, so for $j \in [m+1, n]$

$$\models post(p(\mathbf{s})) \wedge post(p_{m+1}(\mathbf{s_{m+1}}), \ldots, p_{j-1}(\mathbf{s_{j-1}})) \Rightarrow pre(p(\mathbf{s_j})),$$

and thus by (7) for $j \in [m+1, n]$

$$\models post(p_1(\mathbf{s_1}), p_{j-1}(\mathbf{s_{j-1}})) \Rightarrow pre(p(\mathbf{s_j})).$$

This and (8) imply the claim. □

□

This yields the following conclusions.

Theorem 4.7 *Let P and Q well-asserted and let ξ be an LD-derivation of Q in P. All atoms selected in ξ satisfy their precondition.*

Proof. Note that the first atom of a well-asserted query satisfies its precondition and that a variant of a well-asserted clause is well-asserted. The conclusion now follows by Lemma 4.6. □

Corollary 4.8 *Let P and Q be well-asserted. Then for every computed answer substitution σ, $\models post(Q)\sigma$.*

Proof. Let $Q = p_1(\mathbf{s_1}), \ldots, p_k(\mathbf{s_k})$. Let p be a new relation of arity equal to the sum of the arities of p_1, ..., p_k, say n, and with pre_p and $post_p$ both equal to $post_{p_1}\alpha_1 \wedge \ldots \wedge post_{p_k}\alpha_k$, where each α_i renames the p_i-variables to a new set of p-variables. Then $Q, p(\mathbf{s_1}, \ldots, \mathbf{s_k})$ is a well-asserted query. Now, σ is a computed answer substitution for Q in P iff $p(\mathbf{s_1}, \ldots, \mathbf{s_k})\sigma$ is a selected atom in an LD-derivation of $Q, p(\mathbf{s_1}, \ldots, \mathbf{s_k})$ in P. The conclusion now follows by Theorem 4.7. □

Again, let us show how these results can be applied to specific programs.

Example 4.9 Reconsider the program `quicksort`. We associate with its relations the following specifications:

$$
\begin{array}{ll}
pre^{qs} \equiv ListGae(x_1^{qs}), & post^{qs} \equiv perm(x_1^{qs}, x_2^{qs}), sorted(x_2^{qs}), \\
pre^{app} \equiv ListGae(x_1^{app}, x_2^{app}), & post^{app} \equiv conc(x_1^{app}, x_2^{app}, x_3^{app}), \\
pre^{part} \equiv ListGae(x_2^{part}), & post^{part} \equiv \phi(x_1^{part}, x_2^{part}, x_3^{part}, x_4^{part}), \\
pre^{>} = Gae(x_1^{>}, x_2^{>}), & post^{>} = x_1^{>} < x_2^{>} \\
pre^{\leq} = Gae(x_1^{\leq}, x_2^{\leq}), & post^{\leq} = x_1^{\leq} \leq x_2^{\leq},
\end{array}
$$

where $perm(x, y)$ states that x, y are lists and y is a permutation of x, $sorted(x)$ states that x is a sorted list of gae's, $conc(x, y, z)$ states that x, y, z are lists and z is a concatenation of x and y,

$$
\begin{array}{rl}
\phi(x_1^{part}, x_2^{part}, x_3^{part}, x_4^{part}) \equiv & ListGae(x_3^{part}, x_4^{part}),\ el(x_2^{part}) = el(x_3^{part}) \cup el(x_4^{part}), \\
& \forall x(x \in el(x_3^{part}) \rightarrow x < x_1^{part}),\ \forall x(x \in el(x_4^{part}) \rightarrow x \geq x_1^{part}),
\end{array}
$$

where for a list x, $el(x)$ denotes the set of its elements. It is easy to check that `quicksort` is then well-asserted. Assume now that s is a list of gae's. By Theorem 4.7 we conclude that all atoms selected in the LD-derivations of `qs(s,t)` in `quicksort` satisfy their preconditions. In particular, when these atoms are of the form $u > v$ or $u \leq v$, both u and v are gae's. Thus the LD-derivations of `qs(s,t)` do not end in an error. Moreover, by Corollary 4.8 we conclude that all computed answer substitutions σ are such that $t\sigma$ is a sorted permutation of s. □

Thus, static analysis based on monotonic assertions is sufficient to derive information about the form of atom arguments, before or after their selection. This is sufficient to prove monotonic run time properties of programs.

4.2 Correct asserted programs

To deal with run time properties in general we allow also non-monotonic assertions. The corresponding asserted programs are called correct asserted programs. Correct asserted programs were introduced by Drabent and Małuszyński in [DM88]. Let P be a program. The considered *assertion language* contains now the variables $\cdot p_i$, called *input variables*, and the variables $p \cdot_i$ called *output variables*, for $i \in [1, n]$, for every n-ary relation p of P. We follow [DM88] and denote a query $A_1, \ldots, A_n$ by the clause $goal \leftarrow A_1, \ldots, A_n$, where *goal* is a new relation symbol, and both the precondition and the postcondition for *goal* are assumed to be *true*.

Definition 4.10 (specification) A *specification for an n-ary relation p* is a pair $(pre_p, post_p)$ of assertions, s.t. $vars(pre_p) \subseteq \{\cdot p_1, \ldots, \cdot p_n\}$ and $vars(post_p) \subseteq \{\cdot p_1, \ldots, \cdot p_n, p \cdot_1, \ldots, p \cdot_n\}$. □

An *asserted program* $\mathcal{AP}$ is obtained by specifying a specification for every relation defined in P. In the remaining of this section we adopt the following

Assumption 4.11 *Every considered relation* has a fixed specification associated with it.

The following definition explains when an atom satisfies a precondition and a postcondition.

Definition 4.12 Let $A = p(t_1, \ldots, t_n)$. We denote by $pre(A)$ the assertion $pre_p\alpha$, where $\alpha = \{\cdot p_i/t_i \mid i \in [1, n]\}$, and by $post(A, \sigma)$ the assertion $post_p\beta$, where $\beta = \{\cdot p_i/t_i,\ p \cdot_i /(t_i\sigma) \mid i \in [1, n]\}$.

- We say that A *satisfies its precondition* if $\models pre(A)$.

- We say that (A, σ) *satisfies its postcondition* if $\models post(A, \sigma)$.
- We say that (A, σ) *satisfies its specification* if $\models pre(A)$ and $\models post(A, \sigma)$. □

The following notion of a valuation sequence is central in the definition of correctness of an sc-asserted program.

Definition 4.13 (Valuation Sequence) We say that the sequence $\rho_0, \ldots, \rho_n$ of substitutions is a *valuation sequence* for a clause $H \leftarrow B_1, \ldots, B_n$ and an atomic query A if

1. $vars(A) \cap vars(H \leftarrow B_1, \ldots, B_n) = \emptyset$,
2. $\rho_0 = mgu(A, H)$,
3. there exist $\sigma_1, \ldots, \sigma_n$ s.t. for all $i \in [1, n]$

   $\rho_i = \rho_{i-1}\sigma_i$,

   $dom(\sigma_i) \subseteq vars(B_i\rho_{i-1})$,

   $range(\sigma_i) \cap vars((H \leftarrow B_1, \ldots, B_n)\rho_{i-1}) \subseteq vars(B_i\rho_{i-1})$.

□

The above definition provides an abstraction of the notion of a derivation for the atomic query A, when the clause $H \leftarrow B_1, \ldots, B_n$ is chosen as the first input clause: condition 1. expresses the requirement that input clause and query are disjoint, condition 2. defines ρ_0 as the mgu computed by the unification of A and the head H of the clause; finally condition 3. defines for $i \in [1, n]$, σ_i is an abstraction of a computed answer substitution for $B_{i-1}\rho_{i-1}$. The notion of valuation sequence is used to define the concept correct asserted program.

Definition 4.14 (correct asserted program)

- A clause $H \leftarrow B_1, \ldots, B_n$ is called *correct* if, for every atomic query A that satisfies its precondition and for every valuation sequence $\rho_0, \ldots, \rho_n$ for them, for $j \in [1, n+1]$

$$\models post(B_1\rho_0, \sigma_1) \wedge \ldots \wedge post(B_{j-1}\rho_{j-2}, \sigma_{j-1}) \Rightarrow pre(B_j\rho_{j-1}),$$

  where $pre(B_{n+1}\rho_n) \stackrel{\text{def}}{=} post(A, \rho_n)$.
- An asserted program $\mathcal{AP}$ is called *correct* if every clause of it is.

When no ambiguity arises we say that P is correct instead of $\mathcal{AP}$ is correct. □

Now we show that the notion of a well-asserted program is a special case of the notion of correct asserted program. To this aim the following notions and lemma are useful.

Definition 4.15 (unary form) We say that a specification $(pre_p, post_p)$ is in *unary form* if

$$vars(post_p) \cap \{\cdot p_i \mid i \in [1, n]\} = \emptyset, \tag{9}$$

where n is the arity of p. We say that an asserted program is in *unary form* if every its specification is in unary form. □

In other words, a specification is in unary form if the input variables do not occur in its postcondition. When only unary specifications are considered we have that (A, σ) satisfies its postcondition if $\models post_p\beta$, with $\beta = \{ p \cdot_i /(t_i\sigma) \mid i \in [1, n]\}$. In this case we write $\models post(A\sigma)$. Then for asserted programs in unary form Definition 4.14 can be simplified as follows.

Definition 4.16 (correct asserted program in unary form)

- A clause $H \leftarrow B_1, \ldots, B_n$ is called *correct* if, for every atomic query A that satisfies its precondition and for every valuation sequence $\rho_0, \ldots, \rho_n$ for them, for $j \in [1, n+1]$

$$\models post(B_1\rho_1) \wedge \ldots \wedge post(B_{j-1}\rho_{j-1}) \Rightarrow pre(B_j\rho_{j-1}),$$

where $pre(B_{n+1}\rho_n) \stackrel{\text{def}}{=} post(A\rho_n)$.

- An asserted program in unary form $\mathcal{A}P$ is *correct* if every clause of it is.

□

The following property of monotonic assertions follows directly from the definition.

Lemma 4.17 *The truth of a monotonic assertion is invariant under renaming, i.e. if σ is a renaming of the variables of ϕ then $\models \phi \leftrightarrow \phi\sigma$.* □

Assume now that specifications are monotonic and in unary form. We introduce the map u that identifies the input variables with the relative output variables. Let $\alpha = \{\cdot p_i/x_i^p,\ p \cdot_i /x_i^p \mid i \in [1,n]\}$: then

$$u(pre_p, post_p) = (pre_p\alpha, post_p\alpha).$$

Notice that u is a bijection from (monotonic and in unary form) specifications $(pre_p, post_p)$ to specifications $(pre^p, post^p)$ used to define well-asserted programs. Then we can denote $u(pre_p, post_p)$ by $(pre^p, post^p)$.

Theorem 4.18 *The notion of well-asserted program is a special case of the notion of correct asserted program.*

Proof. Let $\mathcal{A}P$ be a correct asserted program in unary form and with monotonic specifications. Let $\mathcal{A}'P$ be the asserted program obtained by replacing every specification $(pre_p, post_p)$ of $\mathcal{A}P$ with $u(pre_p, post_p)$. We show that $\mathcal{A}P$ is correct iff $\mathcal{A}'P$ is well-asserted.

Suppose $\mathcal{A}P$ correct. Let $c = H \leftarrow B_1, \ldots, B_n$ be a clause of P. We prove that c is well-asserted. We have to show that for all $i \in [1, n+1]$

$$\models pre(H) \wedge post(B_1, \ldots, B_{i-1}) \Rightarrow pre(B_i),$$

where $pre(B_{i+1}) \stackrel{\text{def}}{=} post(H)$.

Fix an arbitrary $i \in [1, n+1]$. Let α be s.t.

$$\models pre(H)\alpha \tag{10}$$

and for all $j \in [1, i-1]$

$$\models post(B_j)\alpha. \tag{11}$$

We show that $pre(B_i)\alpha$ is true. Let A be the query $H\alpha$. Consider the sequence $\rho_0, \ldots, \rho_n$, where $\rho_0 = \alpha_{|vars(H)}$ and $\rho_i = \alpha_{|vars(B_i)}$, for $i \in [1, n]$. By Lemma 4.17 we can assume without loss of generality that $vars(A) \cap vars(H \leftarrow B_1, \ldots, B_n) = \emptyset$. Then it is immediate to check that $\rho_0, \ldots, \rho_n$ is a valuation sequence for A and $H \leftarrow B_1, \ldots, B_n$. By (10) A satisfies its precondition. Since $\mathcal{A}P$ is correct, then from (11) and Definition 4.16 we have that $pre(B_i\alpha)$ is true.

Viceversa suppose that $\mathcal{A}'P$ is well-asserted. Let $H \leftarrow B_1, \ldots, B_n$ be a clause of P and let A be s.t.

$$\models pre(A) \tag{12}$$

Let $\rho_0, \ldots, \rho_n$ be a valuation sequence for A and $H \leftarrow B_1, \ldots, B_n$. We have to show that $\models pre(B_1)\rho_0$ and that, for all $i \in [1, n+1]$, if for every $j \in [1, i-1]$

$$\models post(B_j\rho_j) \tag{13}$$

then $\models pre(B_i\rho_i)$.

By hypothesis

$$\rho_0 = mgu(A, H). \tag{14}$$

Then (12), (14) and $\mathcal{A}'P$ well-asserted imply $\models pre(B_1)\rho_0$.

Now, consider an arbitrary $i \in [1, n+1]$ and assume (13) holds for every $j \in [1, i-1]$. By definition of valuation sequence, for all $j \in [0, i-1]$

$$\rho_i = \rho_j\sigma_{j+1}\ldots\sigma_i. \tag{15}$$

From (5) (i.e. by definition of monotonic assertion) and (12) it follows that $\models pre(A)\rho_0\sigma_1\ldots\sigma_i$. Then from (14) and (15) we have that

$$\models pre(H)\rho_i. \tag{16}$$

Moreover, from (5), (13) and (15) it follows that, for all $j \in [1, i-1]$

$$\models post(B_j)\rho_i. \tag{17}$$

Then $\models pre(B_i\rho_i)$ follows from (16), (17) and $\mathcal{A}'P$ well-asserted. □

The following lemma shows persistence of the notion of being correct.

Lemma 4.19 *An LD-resolvent of a correct asserted query and a correct asserted clause that is variable disjoint with it, is correct.*

Proof. Let $Q = A_1, \ldots, A_m$ be a correct asserted query and let $c = H \leftarrow B_1, \ldots, B_n$ be a disjoint with it correct asserted clause. Let $R = (B_1, \ldots, B_n, A_2, \ldots, A_m)\theta$ be the resolvent of Q and c, where $\theta = mgu(H, A_1)$. Suppose that R is represented by the clause $goal \leftarrow R$. To show that R is correct consider a valuation sequence $\rho_0, \ldots, \rho_{n+m-1}$ for R and $goal$. Since $goal$ has no variables, then ρ_0 is equal to the identity substitution ϵ. Thus we have to check the following conditions:

$$\models pre(B_1\theta), \tag{18}$$

$$\text{for all } k \in [1, n-1]\ (\models post(B_1\theta, \rho_1) \wedge \ldots \wedge post(B_k\theta\rho_{k-1}, \sigma_k) \Rightarrow pre(B_{k+1}\theta\rho_k)), \tag{19}$$

$$\models post(B_1\theta, \rho_1) \wedge \ldots \wedge post(B_n\theta\rho_{n-1}, \sigma_n) \Rightarrow pre(A_2\theta\rho_n), \tag{20}$$

and

$$\text{for all } k \in [2, m]\ (\models post(B_1\theta, \rho_1) \wedge \ldots \wedge post(A_k\theta\rho_{k+n-2}, \sigma_{k+n-1}) \Rightarrow pre(A_{k+1}\theta\rho_{k+n-1})). \tag{21}$$

From Q correct we have that $\models pre(A_1)$. Then the sequence of substitutions $\tau_1 = \rho_0^1, \ldots, \rho_n^1$, where $\rho_k^1 = \theta\rho_k$, for $k \in [0, n]$, is a valuation sequence for A_1 and c; moreover the sequence of substitutions $\tau_2 = \rho_0^2, \ldots, \rho_m^2$, where $\rho_0^2 = \epsilon$ and $\rho_k^2 = \theta\rho_{k+n-1}$ for $k \in [1, m]$, is a valuation sequence for Q. Then (18) and (19) follow from c correct and τ_1 valuation sequence for A_1 and c. Moreover c correct and τ_1 valuation sequence for A_1 and c imply

$$\models post(B_1\theta, \rho_1) \wedge \ldots \wedge post(B_n\theta\rho_{n-1}, \sigma_n) \Rightarrow post(A_1, \theta\rho_n). \tag{22}$$

From Q correct and τ_2 valuation sequence for Q we have that for all $k \in [1, m]$

$$\models post(A_1, \theta\rho_n) \wedge \ldots \wedge post(A_k\theta\rho_{k+n-2}, \sigma_{k+n-1}) \Rightarrow pre(A_k\theta\rho_{k+n-1}). \tag{23}$$

Then (22) and (23) imply (21). □

Theorem 4.20 *Let P be a correct program and Q a correct query. Let ξ be an LD-derivation of Q in P. Then all atoms selected in ξ satisfy their precondition.*

Proof. Note that the first atom of a correct query satisfies its precondition and that a variant of a correct clause is correct. The conclusion now follows by Lemma 4.19. □

This yields the following desired conclusions. Notice that now, due to the non-monotonicity of the assertions, the postcondition of a query $Q = A_1, \ldots, A_n$ is $post(A_n)$.

Corollary 4.21 *Let P be a correct program and let Q be a correct query. Then for every computed answer substitution σ, $\models post(A_n)\sigma$.*

Proof. Let $Q = p_1(\mathbf{s_1}), \ldots, p_k(\mathbf{s_k})$. Let p be a new relation of arity equal to the arity of p_k, say n, and with pre_p and $post_p$ both equal to $post_{p_k}\alpha$, where α renames the input and output variables to a new set of input and output variables. Then $Q, p(\mathbf{s_k})$ is a correct query. Now, σ is a computed answer substitution for Q in P iff $p(\mathbf{s_k})\sigma$ is a selected atom in an LD-derivation of $Q, p(\mathbf{s_k})$ in P. The conclusion now follows by Theorem 4.20. □

These results extend to programs containing some built-in relations. For instance the built-in relation *var* can be characterized by the specification $pre_{var} = true$ and $post_{var} = var(var\cdot) \wedge \cdot var = var\cdot$.

In the following example we show how these results can be applied to specific programs.

Example 4.22 Consider the program `length`

```
length([ ],0) ←.
length([x|y],n) ← length(y,m), n:=m+1.
```

which computes the length of a list. Here the query $x := y$ succeeds if y is a gae whose value t is such that x and t unify; otherwise an error occurs. We associate with *length* and $:=$ the following specifications:

$pre_{length} \equiv List(\cdot length_1), var(\cdot length_2)$, $\quad post_{length} \equiv Gae(length_2\cdot), length_2\cdot = |length_1 \cdot|$,
$pre_{:=} \equiv var(\cdot :=_1), Gae(\cdot :=_2)$ $\quad post_{:=} \equiv Gae(:=_1 \cdot), Gae(:=_2 \cdot), :=_1 \cdot = :=_2 \cdot$

where $|\ |$ is a function from lists to natural numbers defined as follows:
$|[\,]| = 0$, $|[x|y]| = 1 + |y|$.
Note that $pre_{:=}$ is the only non-monotonic assertion used.

It is easy to check that `length` is sc-correct. Assume now that x is a list and n a variable. By Theorem 4.20 we conclude that all atoms selected in the LD-derivations of `length(x,n)` in `length` satisfy their precondition. In particular, when these atoms are of the form $n := m + 1$, n is a variable and m is a gae. Thus the LD-derivations of `length` do not end in an error. Moreover, by Corollary 4.21 we conclude that all computed answer substitutions σ are such that $n\sigma$ is the length of the list $x\sigma$. □

Thus, static analysis based on non-monotonic assertions associated with relations is sufficient to derive information about the form of atom arguments, before or after their selection. This is sufficient to prove run time properties of programs.

4.3 Partial correct programs

In the two methods presented in the previous sections, specifications are associated with the relations defined in the program. To deal with assertions associated with control points, we introduce the notion of partial correct program. This notion is based on the procedural interpretation of a logic program. In this interpretation the meaning of a clause $H \leftarrow B_1, \ldots, B_n$ is: *to solve H solve B_1, then B_2, ..., then B_n*. In this approach we analyze the values of the variables of the program when the computation reaches the *program point* before or after the execution of an atom in the body of a clause. This can be done by decorating the clauses with assertions $H \leftarrow \{I_0\}B_1\{I_1\}\ldots B_n\{I_n\}$, that describe the possible values of the variables of the clause when the execution reaches the relative program point. Colussi and Marchiori in [CM91] proposed a method based on this approach. The assertions considered contain the variables of the program and all their renamings. It is assumed that assertions are semantically invariant w.r.t. renaming, i.e. $\models \phi \leftrightarrow \phi\sigma$, for every assertion ϕ and renaming σ of $vars(\phi)$. Let P be a program.

Definition 4.23 (asserted program) An *asserted clause* is defined as

$$H \leftarrow \{I_0\}B_1\{I_1\}\ldots B_n\{I_n\},$$

where $H \leftarrow B_1,\ldots,B_n$ is a clause of P and $I_0, \ldots, I_n$ are assertions. The formulas $\{I_{i-1}\}B_i\{I_i\}$, $i \in [1,n]$, are called *specifications*. An *asserted program* $\mathcal{AP}$ is a set of asserted clauses, one for every clause of P. □

Informally, for an asserted program to be partially correct, the assertions $I_0, I_1, \ldots, I_n$ must be proven to be global invariants, i.e. invariant with respect to the multiple definitions and to the recursion. To this end, a predicate relation is associated with the unification. Unification is expressed through a set of sets $\mathcal{U}$ of terms/atoms. A unifier for $\mathcal{U}$ is a substitution that reduces every set of $\mathcal{U}$ to a singleton. β is an *mgu* of $\mathcal{U}$ if it is a unifier and $\alpha = \beta\gamma$, for every other unifier α of $\mathcal{U}$, for some substitution γ. $mgu(\mathcal{U})$ denotes the set of idempotent *mgu*'s of $\mathcal{U}$.

Definition 4.24 (The relation $\{p\}\mathcal{U}\{q\}$) Let $\mathcal{U}$ be a set of sets of terms/atoms and let p and q be assertions. Then $\{p\}\mathcal{U}\{q\}$ holds if and only if for all substitution α such that $p\alpha$ is true and there exists $\mu \in mgu(\mathcal{U}\alpha)$, we have that $q\alpha\mu$ is true.

In [CM91] a proof-system for $\{p\}\mathcal{U}\{q\}$ is given.

The following definition of *matches* is central in the concept of partial correct program. First, we give some notation. With a substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$, we associate the set of sets of terms $\mathcal{E}_\theta = \{\{x_1,t_1\},\ldots,\{x_n,t_n\}\}$. For a specification $spec = \{pre\}A\{post\}$, those free variables that do not occur in A are called *auxiliary variables*, denoted $aux(spec)$. More generally we write $aux_V(E)$ to denote the set of free variables that occur in V but not in E.

Definition 4.25 (Matches) Let $spec = \{pre\}A\{post\}$ be a specification and let $\mathcal{A}c = H \leftarrow \{I_0\}B_1$ $\{I_1\}\ldots B_n\{I_n\}$ an asserted clause. We say that *spec matches* $\mathcal{A}c$ if there exist a variant $\mathcal{A}c'$ of $\mathcal{A}c$, two disjoint sets of variables X, Y and a substitution ρ such that

(i) $X \supseteq vars(spec)$,

(ii) $Y \supseteq vars(\mathcal{A}c')$,

(iii) $dom(\rho) \subseteq aux_Y(c')$ and

(iv) $range(\rho) \supseteq aux_X(A)$

and such that

$$\{pre \wedge\ free(Y; X \cup Y)\}\ \mathcal{U}\ \{I'_0\} \qquad DOWN$$

$$\{I'_n \wedge\ free(X; X \cup Y)\}\ \mathcal{U}\ \{post\} \qquad UP$$

where

$$\mathcal{U} = \{\{A, H'\}\} \cup \mathcal{E}_\rho,$$

and $free(X;Z)$ holds when X and Z are sets of variables s.t. every variable x in X is disjoint from all variables $z \neq x$ of Z. Formally

$$free(X;Z)\alpha \equiv \forall x \in X, z \in Z(var(x\alpha) \wedge var(z\alpha) \wedge (x \neq z \Rightarrow x\alpha \neq z\alpha)).$$

□

We say that a specification *spec* matches the asserted program $\mathcal{AP}$ if *spec* matches every asserted clause of $\mathcal{AP}$.

Definition 4.26 (partial correct program) Let $\mathcal{AP}$ be an asserted program. We say that $\mathcal{AP}$ is *partially correct* if every its specification matches $\mathcal{AP}$. When no ambiguity arises, we say P partial correct instead of $\mathcal{AP}$ partial correct. □

$inst_{A_1,A_2}(x,y)\alpha$	$\equiv$	$(x\alpha = A_1\rho \wedge \rho = mgu(y\alpha, A_2))$;
$inst(x,y,w,z)\alpha$	$\equiv$	$(x\alpha = y\alpha\sigma \wedge \sigma = mgu(w\alpha, z\alpha))$;
$incl(x,y,z)\alpha$	$\equiv$	$(vars(x\alpha) \cap vars(y\alpha) \subseteq vars(z\alpha))$;
$(x \succeq y)\alpha$	$\equiv$	$(x\alpha = y\alpha\beta)$;
$Disj_t(x)\alpha$	$\equiv$	$(vars(x\alpha) \cap vars(t) = \emptyset)$.

Table 1: Auxiliary relations

Notice that, while in the verification condition of the two previous methods the clauses of the program are examined independently, here the verification condition is global, i.e. it relates a specification of an asserted clause with all the clauses of the program. This is needed to guarantee that the assertions decorating the program are *global invariants.*

Now we show that the notion of partial correct program is a generalization of the notion of correct program. To this aim we introduce first suitable assertions, to describe the verification condition of a correct program. Next, given an asserted program $\mathcal{A}P$, we add a suitable asserted clause to $\mathcal{A}P$ s.t. the resulting program P' coincides with P when restricted to the relations defined in P. We show that the resulting asserted program $\mathcal{A}'P'$ is partial correct iff $\mathcal{A}P$ is correct.

Consider an atom A and a clause $c = H \leftarrow B_1, \ldots, B_n$. To define an assertion $VS_c(A)$ s.t. $VS_c(A)\alpha$ characterizes the valuation sequences for $A\alpha$ and c, we use the relations given in Table 1.

Definition 4.27 (The assertion $VS_c(A)$) Let A be an atom and let $c = H \leftarrow B_1, \ldots, B_n$. Then the assertion $VS_c(A)$ is defined as follows.

$$VS_c(A) = (Disj_c(A) \wedge Init \wedge (Inst_1 \wedge Rel_1) \wedge \ldots \wedge (Inst_n \wedge Rel_n)),$$

where

$$Init = (inst_{H,H}(x_0^0, A) \wedge inst_{B_1,H}(x_1^0, A) \ldots \wedge inst_{B_n,H}(x_n^0, A)),$$

$$Inst_i = (inst(x_0^i, x_0^{i-1}, x_i^{i-1}, y_i^{i-1}) \wedge \ldots \wedge inst(x_n^i, x_n^{i-1}, x_i^{i-1}, y_i^{i-1})),$$

$$Rel_i = (y_i^{i-1} \succeq x_i^{i-1} \wedge incl(y_i^{i-1}, (x_0^{i-1}, \ldots, x_n^{i-1}), x_i^{i-1})).$$

□

The following lemma clarifies the role of the x_i^k's in $VS_c(A)$.

Lemma 4.28 *Let $c = H \leftarrow B_1, \ldots, B_n$. For every substitution α, if $\models VS_c(A)\alpha$ then for $i \in [1,n]$ and for $k \in [0,n]$ we have that*

$$x_k^i\alpha = B_k\rho_0\sigma_1 \ldots \sigma_i,$$

where $B_0 \stackrel{\text{def}}{=} H$, $\rho_0 = mgu(A\alpha, H)$ and $\sigma_i = mgu(x_i^{i-1}\alpha, y_i^{i-1}\alpha)$.

Proof. Let α be s.t. $\models VS_c(A)\alpha$. Then for all $i \in [1,n]$, from $\models Inst_i\alpha$ it follows that, for all $k \in [0,n]$,

$$x_k^i\alpha = x_k^{i-1}\alpha\sigma_i, \tag{24}$$

where

$$\sigma_i = mgu(x_i^{i-1}\alpha, y_i^{i-1}\alpha). \tag{25}$$

Moreover, from $\models Init\alpha$, it follows that for all $k \in [0,n]$

$$x_k^0\alpha = B_k\rho_0, \tag{26}$$

where

$$\rho_0 = mgu(H, A\alpha). \tag{27}$$

Now, fix an arbitrary k in $[0,n]$. We prove by induction on $i \in [1,n]$ that

$$x_k^i\alpha = B_k\rho_0\sigma_1 \ldots \sigma_i. \tag{28}$$

The base case follows from (24) and (26). Now suppose (28) holds for $i-1$. From (24) it follows that $x_k^i\alpha = x_k^{i-1}\alpha\sigma_i$. Then from the inductive hypothesis applied to $x_k^{i-1}\alpha$ we obtain (28).

□

Now we show that Definition 4.27 characterizes the intended meaning of $VS_c(A)$, i.e. $VS_c(A)$ describes the valuation sequences of c and intances $A\alpha$ of A.

Theorem 4.29 *Let $c = H \leftarrow B_1, \ldots, B_n$. Then for every substitution α,*

(i) if $\models VS_c(A)\alpha$ then there exists a valuation sequence $\rho_0, \ldots, \rho_n$ for $A\alpha$ and c,

(ii) if $\rho_0, \ldots, \rho_n$ is a valuation sequence for $A\alpha$ and c then there exists α' s.t. $\alpha'_{|vars(A)} = \alpha$ and $\models VS_c(A)\alpha'$.

Proof.
(*i*) Let α s.t. $\models VS_c(A)\alpha$. Then for all $i \in [1,n]$, from $\models Rel_i\alpha$ it follows that there exists β_i with $dom(\beta_i) \subseteq vars(x_i^{i-1}\alpha)$ s.t. $y_i^{i-1}\alpha = x_i^{i-1}\alpha\beta_i$, and from (24) we obtain $\beta_i = \sigma_i$. Therefore

$$y_i^{i-1}\alpha = x_i^{i-1}\alpha\sigma_i \tag{29}$$

and

$$dom(\sigma_i) \subseteq vars(x_i^{i-1}\alpha). \tag{30}$$

We show that the required valuation sequence for $A\alpha$ and c is the sequence $\tau = \rho_0, \ldots, \rho_n$ of substitutions, with $\rho_i = \rho_0\sigma_1 \ldots \sigma_i$, where ρ_0 and σ_i are defined respectively in (27) and (25). We have to check that τ satisfies 1. - 3. of the Definition 4.13 of valuation sequence. From $\models Disj_c(A)\alpha$ and from (27) it follows that τ satisfies 1. and 2. It remains to check the three conditions of 3. The first condition $\rho_i = \rho_{i-1}\sigma_i$ holds by construction. Lemma 4.28 and 2. imply $vars(x_i^{i-1}\alpha) = vars(B_i\rho_{i-1})$. Hence from (30) it follows the second condition $dom(\sigma_i) \subseteq vars(B_i\rho_{i-1})$ of 3. Now (30) and (29) imply

$$range(\sigma_i) \subseteq vars(y_i^{i-1}\alpha). \tag{31}$$

Since $\models Rel_i\alpha$ implies $(vars(y_i^{i-1}\alpha) \cap vars((x_0^{i-1}\alpha, \ldots, x_n^{i-1}\alpha))) \subseteq vars(x_i^{i-1}\alpha)$, then from Lemma 4.28

$$(vars(y_i^{i-1}\alpha) \cap vars((H \leftarrow B_1, \ldots, B_n)\rho_{i-1}) \subseteq vars(a_i\rho_{i-1}). \tag{32}$$

From (31) and (32) follows the last condition $range(\sigma_i) \cap vars((H \leftarrow B_1, \ldots, B_n)\rho_{i-1}) \subseteq vars(B_i\rho_{i-1})$ of 3.

(*ii*) Suppose $\tau = \rho_0, \ldots, \rho_n$ is a valuation sequence for $A\alpha$ and c. Then there are substitutions $\sigma_1, \ldots, \sigma_n$ satisfying 3. of Definition 4.13. We choose α' s.t.

$$x\alpha' = \begin{cases} x\alpha & \text{if } x \in vars(A) \\ x_i^{i-1}\rho_0\sigma_1 \ldots \sigma_i & \text{if } x = y_i^{i-1} \\ B_k\rho_0\sigma_1 \ldots \sigma_i & \text{if } x = x_k^i \end{cases}$$

It is easy to check that α' is well defined and that $\models VS_c(A)\alpha'$. □

We can now prove the desired result. For an assertion I we denote by $\forall I$ its universal closure.

Theorem 4.30 *The notion of correct program is a special case of the notion of partial correct program.*

Proof. Let $\mathcal{A}P$ be a program asserted with specifications as in Definition 4.10 and let $\mathcal{A}'P'$ be the asserted program defined as follows.

$$P' = P \cup \{c_p \mid p \text{ defined in } P\},$$

where $c_p = goal_p \leftarrow p(\mathbf{x})$, with $goal_p$ new distinct relation symbols.

For every clause c of P let

$$\mathcal{A}'c = H \leftarrow \{I_c\}B_1, \ldots, \{I_c\}B_n\{I_c\},$$

where I_c is the assertion

$$\forall(pre(x_c) \wedge VS_c(x_c) \Rightarrow (pre_q(x_1^0) \wedge (\wedge_{i\in[1,n]}(post_{q_1}(x_1^0, x_1^1) \wedge \ldots \wedge post_{q_i}(x_i^{i-1}, x_i^i) \Rightarrow pre_{q_{i+1}}(x_{i+1}^{i+1})))),$$

where $pre_{q_{n+1}}(x_{n+1}^{n+1}) = post_q(x_c, x_0^n)$ and q, q_1, ..., q_n are the relation symbols of H, B_1, ..., B_n, respectively. For every clause c_p of P' let

$$\mathcal{A}'c_p = goal_p \leftarrow \{true\}\, p(\mathbf{x})\, \{true\}.$$

We prove that $\mathcal{A}'P'$ is partial correct iff $\mathcal{A}P$ is correct. Suppose that $\mathcal{A}'P'$ is partial correct. Then $\{true\}\ p(\mathbf{x})\{true\}$ matches $\mathcal{A}'c$. Then from $DOWN$, since I_c is a closed formula, we have that $\models I_c$. Then from Theorem 4.29 it follows that $\mathcal{A}P$ is correct.

Viceversa suppose $\mathcal{A}P$ correct. Then from Theorem 4.29 $\models I_c$ holds. Thus, every specification in the body of $\mathcal{A}'P$ matches $\mathcal{A}'P'$. Moreover the specification in the body of $\mathcal{A}c_p$ matches $\mathcal{A}'P'$, because its pre- and postconditon are both *true*. Hence $\mathcal{A}'P'$ is partial correct. □

We now show persistence of being partial correct, in the following sense. First, we introduce some useful terminology.

For an asserted query $\mathcal{A}Q = \{R_0\}\mathbf{A_n}\{R_n\}$ we call R_0 and R_n *precondition of* $\mathcal{A}Q$ and *postcondition of* $\mathcal{A}Q$, written $pre(\mathcal{A}Q)$ and $post(\mathcal{A}Q)$, respectively. We say that $\mathcal{A}Q$ matches the asserted program $\mathcal{A}P$ if every specification of $\mathcal{A}Q$ matches $\mathcal{A}P$. We call a sequence of asserted queries $\mathcal{A}Q_1, \ldots, \mathcal{A}Q_n$ *compound asserted query*. We say that a compound asserted query matches $\mathcal{A}P$ if every its asserted query matches $\mathcal{A}P$. We introduce the notion of asserted resolvent of a compound asserted query and an asserted clause.

Definition 4.31 Let $\mathcal{A}Q_1, \ldots, \mathcal{A}Q_k$ be a compound asserted query. Let $\mathcal{A}c = H \leftarrow \{I_0\}\mathbf{B_m}\{I_m\}$ be an asserted clause disjoint with all the $\mathcal{A}Q_i$'s. Suppose $\mathcal{A}Q_1 = \{R_0\}\mathbf{A_n}\{R_n\}$ and suppose $mgu(H, A_1)$ exists, say θ. Then the *asserted resolvent of* $\mathcal{A}Q_1, \ldots, \mathcal{A}Q_k$ *and* $\mathcal{A}c$, $\mathcal{A}R$, is defined as follows.

(i) If $m = 0$ then

$$\mathcal{A}R = (\{R_1\}A_2 \ldots \{R_{n-1}\}A_n\{R_n\}, \mathcal{A}Q_2, \ldots, \mathcal{A}Q_k)\theta.$$

In this case we say that $\mathcal{A}R$ *satisfies its precondition* if $R_1\theta$ is true.

(ii) If $m > 0$ then

$$\mathcal{A}R = (\{I_0\}\mathbf{B_m}\{I_m\}, \{R_1\}A_2 \ldots \{R_{n-1}\}A_n\{R_n\}, \mathcal{A}Q_2, \ldots, \mathcal{A}Q_k)\theta.$$

In this case we say that $\mathcal{A}R$ *satisfies its precondition* if $\exists I_0\theta$ is true, where $\exists$ quantifies over $aux_{\mathcal{A}c\theta}(c\theta)$.

□

We say that a *compound asserted query* $\mathcal{A}Q_1, \ldots, \mathcal{A}Q_k$ *satisfies its precondition* if $pre(\mathcal{A}Q_1)$ is true. Thus we can consider compound asserted queries to be resolvents of type (i). The notion of asserted derivation is defined in the natural way as follows.

Definition 4.32 Let $\mathcal{AQ}$ be a asserted query and let $\mathcal{AP}$ be an asserted program. Then a *asserted LD-derivation of* $\mathcal{AQ}$ in $\mathcal{AP}$ is a maximal sequence $\mathcal{AQ}_0, \mathcal{AQ}_1, \ldots$ of asserted queries, s.t. $\mathcal{AQ}_0 = \mathcal{AQ}$, and every $\mathcal{AQ}_{i+1}$ is an asserted resolvent of $\mathcal{AQ}_i$ and a (variant) $\mathcal{A}c$ of an asserted clause in $\mathcal{AP}$, where $\mathcal{A}c$ is disjoint with $\mathcal{AQ}$ and with all asserted clauses used to derive the previous asserted resolvents. □

The following lemmas are useful.

Lemma 4.33 *([CM91]) If a specification spec matches an asserted clause $\mathcal{A}c$, then every instance spec' of spec, s.t. no free variable of spec occurs bounded in spec', matches $\mathcal{A}c$.* □

We can now prove the persistence of the notion being partial correct.

Lemma 4.34 *Let $\mathcal{AP}$ be a partial correct program, let $\mathcal{AQ}$ be an asserted query, let $\mathcal{A}\xi$ be an asserted LD-derivation of $\mathcal{AQ}$ in $\mathcal{AP}$. Let $\mathcal{AR} = \mathcal{AQ}_1, \ldots, \mathcal{AQ}_k$ be an asserted resolvent in $\mathcal{A}\xi$. Suppose that*

- *$\mathcal{AR}$ matches $\mathcal{AP}$,*
- *$\mathcal{AR}$ satisfies its precondition.*

Let $\mathcal{AR}'$ be the asserted resolvent of $\mathcal{AR}$ in $\mathcal{A}\xi$. Then

1. *$\mathcal{AR}'$ matches $\mathcal{AP}$,*
2. *$\mathcal{AR}'$ satisfies its precondition.*

Proof.
1. Immediate by definition of asserted resolvent and Lemma 4.33.

2. Let $\mathcal{AQ}_1 = \{R_0\}\mathbf{A_n}\{R_n\}$ and let $\mathcal{A}c' = H \leftarrow \{I_0\}\mathbf{B_m}\{I_m\}$ the asserted clause used to obtain the resolvent $\mathcal{AR}'$. Then

$$\mathcal{AR}' = (\{I_0\}\mathbf{B_m}\{I_m\}, \{R_1\}A_2 \ldots A_n\{R_n\}, \mathcal{AQ}_2, \ldots, \mathcal{AQ}_k)\theta,$$

where $\theta = mgu(H, A_1)$. Since $\mathcal{AR}$ satisfies its precondition, then for a suitable α we have that $pre(\mathcal{AQ}_1)\alpha$ is true, where

$$dom(\alpha) \subseteq aux_{\mathcal{AR}}(R). \tag{33}$$

Notice that if $\mathcal{AR}$ is of type (i) then $\alpha = \epsilon$. It is not restrictive to assume that $range(\alpha) \cap vars(\mathcal{A}c') = \emptyset$. By Lemma 4.33 we have that $spec = (\{R_0\}A_1\{R_1\})\alpha$ matches $\mathcal{A}c'$. Then there exist X, Y and ρ s.t. conditions (i)-(iv) of the definition are satisfied. Let $\sigma = (\rho\alpha)_{|dom(\rho)}$. From (33) it follows that $A_1\alpha = A_1$. Let $\mathcal{U} = \{\{A_1, H\}\} \cup \mathcal{E}_\rho$. Then it can be shown that $\theta\sigma \in mgu(\mathcal{U})$. Then from DOWN

$$I_0\theta\sigma \tag{34}$$

is true. We distinguish two cases.

If $m = 0$ then we have to show that $R_1\theta$ is true. Let $\beta = (\theta\sigma)_{|vars(\mathcal{A}c')}$. Then $(I_0 \wedge free(X; X \cup Y))\beta$ holds, so from UP we have that $R_1\beta\theta_{|vars(A_1)}\sigma$ holds. Then from $dom(\rho) \cap (vars(spec) \cup vars(c')) = \emptyset$ and from *spec* and $\mathcal{A}c'$ disjoint, it follows that $R_1\beta\theta_{|vars(A_1)}\sigma = R_1\theta$. Then $R_1\theta$ is true.

If $m > 0$ then from (34) it follows that $\exists I_0\theta$ is true. □

Theorem 4.35 *Let $\mathcal{AP}$ be a partial correct program and let $\mathcal{AQ}$ be an asserted query s.t. $\mathcal{AQ}$ satisfies its precondition and $\mathcal{AQ}$ matches $\mathcal{AP}$. Let $\mathcal{A}\xi$ be an asserted derivation of $\mathcal{AQ}$ in $\mathcal{AP}$. Then all resolvents of $\mathcal{A}\xi$ satisfy their preconditions.*

Proof. Note that $\mathcal{AQ}$ satisfies its precondition and that a variant of a partial correct program is partial correct. The conclusion now follows by Lemma 4.34. □

This yields the following conclusions.

Corollary 4.36 *Let $\mathcal{AP}$ be a partial correct program and let $\mathcal{AQ}$ be an asserted query s.t. $\mathcal{AQ}$ matches $\mathcal{AP}$. Let α s.t. $pre(\mathcal{AQ})\alpha$ is true. Then for every computed answer substitution σ of $Q\alpha$ in $P \models post(\mathcal{AQ})\alpha\sigma$.*

Proof. Let $\mathcal{AQ}\alpha = \{R_0\}\mathbf{An}\{R_n\}$. Let p be a new relation symbol and let $\mathcal{AQ}' = \{R_0\}\mathbf{An}\{R_n\}p\{R_n\}$. Then $\mathcal{AQ}'$ satisfies its precondition and matches $\mathcal{AP}$. Now, σ is a computed answer substitution for $Q\alpha$ in P iff $\{R_n\sigma\}p\{R_n\sigma\}$ is an asserted resolvent of type (ii) in a asserted derivation of $\mathcal{AQ}'$ in $\mathcal{AP}$. The conclusion now follows by Theorem 4.35. □

Example 4.37 Reconsider the program `length`, asserted as follows.

`goal` $\leftarrow$ $\{var(n), List(x)\}$ `length(x,n)` $\{List(x), n = |x|\}$.
`length([ ],0)` $\leftarrow$ $\{I_1\}$.
`length([x|y],n)` $\leftarrow$ $\{I_2\}$ `length(y,m)` $\{I_3\}$ `n:=m+1` $\{I_4\}$.

where the assertions I_i are defined below:

$I_1 \equiv true$,
$I_2 \equiv var(n), List(y), var(m)$,
$I_3 \equiv Gae(m), var(n), List(y), m = |y|$,
$I_4 \equiv Gae(n), n = |[x|y]|$

It is easy to verify that $\mathcal{A}$`length` is partial correct. Then by Theorem 4.35 we conclude that all the asserted resolvents in the asserted derivations of $\{var(n), List(x)\}$ `length(x,n)`$\{List(x), n = |x|\}$ in $\mathcal{A}$`length` satisfy their precondition. In particular, when the lefmost atom of an asserted resolvent is of the form $n := m + 1$, then n is a variable and m is a gae. Thus the LD-derivations of `length` do not end in an error. Moreover, by Corollary 4.36 we conclude that all computed answer substitutions σ are s.t. $n\sigma$ is the length of the list $x\sigma$.

□

Thus, static analysis based on non-monotonic assertions associated with control points is sufficient to derive information about the value of atom variables, before or after their selection. This is sufficient to prove run time properties of programs.

We conclude by studying partial correct programs that use only monotonic assertions.

4.4 Partial correct programs with monotonic assertions

Here we consider partial correct programs that use only monotonic assertions. We show that the definition of partial correctness becomes simpler and give a direct proof that the notion of well-asserted program is a special case of the notion of partial correct program.

With the set of sets of terms/atoms $\mathcal{U} = \{\{t_1^1, \ldots, t_{n1}^1\}, \ldots, \{t_1^m, \ldots, t_{nm}^m\}\}$ we associate the assertion

$$\mathcal{AU} = (t_1^1 = \ldots = t_{n1}^1) \wedge \ldots \wedge (t_1^m = \ldots = t_{nm}^m).$$

Then the Definition 4.25 of *matches* can be simplified as follows.

Lemma 4.38 *Let $\mathcal{AP}$ be an asserted program. Assume that the assertions occurring in $\mathcal{AP}$ are monotonic. Then DOWN and UP of Definition 4.25 are equivalent respectively to*

$(pre \wedge \mathcal{AU}) \Rightarrow R_0'$ *DOWN'*

$(R_n' \wedge \mathcal{AU}) \Rightarrow post$ *UP'.*

Proof. Suppose DOWN' holds. Let α s.t. $\models (pre \wedge free(Y, X \cup Y))\alpha$ and let $\beta \in mgu(\mathcal{U}\alpha)$. We have to prove $\models R_0'\alpha\beta$. The definition of $\mathcal{AU}$ and pre monotonic imply $\models (pre \wedge \mathcal{AU})\alpha\beta$. Since DOWN' holds, then $\models R_0'\alpha\beta$.

Viceversa suppose DOWN holds. Let α s.t. $\models (pre \wedge \mathcal{A}U)\alpha$. We have to prove $\models R'_0\alpha$. Let $\rho = \{v_1/t_{v1}, \ldots, v_k/t_{vk}\}$. Then for all $i \in [1,k]$ we have

$$v_i\alpha = t_{vi}\alpha. \tag{35}$$

Let σ be a renaming of Y s.t. $free(Y\sigma, Y\sigma \cup X\alpha)$ holds. We define the substitution α' as follows:

$$x\alpha' = \begin{cases} x\alpha & \text{if } x \in vars(X) \\ x\sigma & \text{if } x \in vars(Y) \end{cases}$$

α' is well defined because X and Y are disjoint. Moreover it is easy to check that $\beta = \{(v_i\alpha')/(t_{vi}\alpha)\}$ is in $mgu(\mathcal{U}\alpha')$. Since DOWN holds, then $\models R'_0\alpha'\beta$. From (35) it follows that $R'_0\alpha = R'_0\alpha'\beta$. Hence $\models R'_0\alpha$. This concludes the proof that DOWN is equivalent to DOWN'.

The proof for UP and UP' is similar. Suppose UP' holds. Let α s.t. $\models (R'_n \wedge free(X, X \cup Y))\alpha$ and let $\beta \in mgu(\mathcal{U}\alpha)$. We have to prove $\models post\alpha\beta$. The definition of $\mathcal{A}U$ and R'_n monotonic imply $\models (R'_n \wedge \mathcal{A}U)\alpha\beta$. Since UP' holds, then $\models post\alpha\beta$.

Viceversa suppose UP holds. Let α s.t. $\models (R'_n \wedge \mathcal{A}U)\alpha$. We have to prove $\models post\alpha$. Let $\rho = \{v_1/t_{v1}, \ldots, v_k/t_{vk}\}$. Then for all $i \in [1,k]$ we have

$$v_i\alpha = t_{vi}\alpha. \tag{36}$$

Let σ be a renaming of Y s.t. $free(Y\sigma, Y\sigma \cup X\alpha)$ holds. We define the substitution α' as follows:

$$x\alpha' = \begin{cases} x\alpha & \text{if } x \in vars(X) \\ x\sigma & \text{if } x \in vars(Y) \end{cases}$$

α' is well defined because X and Y are disjoint. Let $\beta \in mgu(\mathcal{U}\alpha')$. Since UP holds, then $\models post\alpha'\beta$. It follows from the definition of α' that $mgu(\mathcal{U}\alpha') = mgu(\mathcal{E}_\rho\alpha')$. Hence from (36) $t_{vi}\alpha'\beta = t_{vi}\alpha$. Then $post\alpha'\beta = post\alpha$. Hence $\models post\alpha$. □

Theorem 4.39 *The notion of a well-asserted program is a special case of the notion of a partial correct program.*

Proof. Let $\mathcal{A}P$ be an program asserted with specifications as in Definition 4.1. Consider the asserted program $\mathcal{A}'P'$ s.t.

$$P' = P \cup \{c_p \mid p \text{ defined in } P\},$$

where $c_p = goal_p \leftarrow p(\mathbf{x})$. For every clause of P let

$$\mathcal{A}c = H \leftarrow \{I_c\}B_1 \ldots B_n\{I_c\},$$

where I_c is the assertion

$$\forall(pre(H) \wedge post(B_1, \ldots, B_i) \Rightarrow pre(B_{i+1}))),$$

and let

$$\mathcal{A}c_p = goal_p \leftarrow \{true\}p(\mathbf{x})\{true\}.$$

We prove that $\mathcal{A}'P'$ is partial correct iff $\mathcal{A}P$ is well-asserted. Suppose that $\mathcal{A}'P'$ is partial correct. Then the specification in the body of $\mathcal{A}c_p$ matches $\mathcal{A}c$: from DOWN and I_c closed formula it follows that $\models I_c$, which implies that c is well-asserted.

Viceversa suppose that $\mathcal{A}P$ is well-asserted. Then $\models I_c$ holds. So every specification in the body of $\mathcal{A}'P$ matches $\mathcal{A}'P'$. Moreover the specification in the body of $\mathcal{A}c_p$ matches $\mathcal{A}'P'$, because its pre- and postconditon are both *true*. Hence $\mathcal{A}'P'$ is partial correct. □

References

[Apt90] K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–574. Elsevier, 1990. Vol. B.

[BC89] A. Bossi and N. Cocco. Verifying correctness of logic programs. In *Proceedings of Tapsoft '89*, pages 96–110, 1989.

[BLR92] F. Bronsard, T.K. Lakshman, and U.S. Reddy. A framework of directionality for proving termination of logic programs. In K.R. Apt, editor, *Proc. of the Joint International Conference and Symposium on Logic Programming*, pages 321–335. MIT Press, 1992.

[CM91] L. Colussi and E. Marchiori. Proving correctness of logic programs using axiomatic semantics. In *Proceedings of the Eight International Conference on Logic Programming*, pages 629–644. The MIT Press, 1991.

[DM85] P. Dembinski and J. Maluszynski. AND-parallelism with intelligent backtracking for annotated logic programs. In *Proceedings of the International Symposium on Logic Programming*, pages 29–38, Boston, 1985.

[DM88] W. Drabent and J. Maluszynski. Inductive assertion method for logic programs. *Theoretical Computer Science*, 59(1):133–155, 1988.

[Dra87] W. Drabent. Do logic programs resemble programs in conventional languages? In *Proc. of the Joint International Symposium on Logic Programming*, pages 389–396. IEEE Computer Society, 1987.

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

[Mel81] C.S. Mellish. The automatic generation of mode declarations for prolog programs. Technical report, Department of Artificial Intelligence, Univ. of Edinburgh, 1981. DAI Research Paper 163.

[Red84] U. S. Reddy. Transformation of logic programs into functional programs. In *International Symposium on Logic Programming*, pages 187–198. IEEE Computer Society, 1984.

[Red86] U.S. Reddy. On the relationship between logic and functional languages. In D. DeGroot and G. Lindstrom, editors, *Functional and Logic Programming*, pages 3–36. Prentice-Hall, 1986.

[Ros91] D.A. Rosenblueth. Using program transformation to obtain methods for eliminating backtracking in fixed-mode logic programs. Technical Report 7, Universidad Nacional Autonoma de Mexico, Instituto de Investigaciones en Matematicas Aplicadas y en Sistemas, 1991.