Università di Pisa
Dipartimento di Informatica

Technical Report : TR-96-24

Modular Verification of Logic Programs

Dino Pedreschi, Salvatore Ruggieri

June 18, 1996

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

Modular Verification of Logic Programs

Dino Pedreschi and **Salvatore Ruggieri**

Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy
e-mail: {pedre, ruggieri}@di.unipi.it

June 18, 1996

Abstract

Recentely, in [Rug94, PR96] a new approach to verification of logic and Prolog programs has been proposed, whose main advantage is the possibility to reason on different properties in a unified framework. In this paper, we show an equivalent formulation of that proof method which is well-suited for modular program verification. The notion of modularity taken into account is based on stratification. We show that the proofs of correctness of a program P and a program Q can be re-used in the proof of correctness of $P \cup Q$ with small additional efforts. The main advantage consists in the fact that the proofs for P and Q are developed in a fully independent way.

Keywords: Verification of logic and Prolog programs, Systematic program development, Modular programming.

1 Introduction

Logic programming (and Prolog) is advertised as a *declarative* language, in the sense that *specifications*, when written in an appropriate syntax, can be directly used as *programs*. This ideal situation, however, is often unrealistic in practice, both for the fact that executable specifications may be extremely inefficient, and for problems related to the implementations of logic programming—programs may fail to terminate or may result in errors when executed with particular strategies.

It is therefore needed to assess the correctness of a logic or Prolog programs with respect to its specification, or *intended interpretation*. Recentely, in [Rug94, PR96] a new approach to verification of logic and Prolog programs has been proposed whose main advantage is the possibility to reason on several properties by means of a unified framework. In this paper, we show an equivalent formulation of that proof method which is well-suited for modular program verification. The notion of modularity taken into account is based on stratification. We show that the proofs of correctness of a program P and a program Q can be re-used in the proof of correctness of $P \cup Q$ with small additional efforts. The main advantage consists of the fact that the proofs for P and Q are developed in a fully independent way.

Notation

A word on terminology is in order. Throughout the paper we use the standard notation of logic programming, as in [Llo87, Apt90], unless specified otherwise. A language is a pair $\langle \Sigma_L, \Pi_L \rangle$ of (not necessarily disjoint) set of function and predicate symbols. To each symbol a non-negative arity is assigned. Ambivalent syntax is allowed, in the sense that function and predicate symbols may overlap [KJ95, Jia94].

We use queries instead of goals and consider a fixed language L in which programs and queries are written. In other words, all the results are *parametric* with respect to L, provided L is rich enough to contain every symbol of the programs and queries under consideration.

For two programs P and Q, we say that P extends Q if no relation symbol defined in P appears in Q. L_P is the language generated by the program P. By M_P^L we denote the least Herbrand model of P with L as the underlying language. M_P is an abbreviation for $M_P^{L_P}$. B_L is the Herbrand base of L. B_P is an abbreviation for B_{L_P}. We denote with $[A]_L$ the set of ground instances on the language L of the atom A. By $A \leftarrow B_1, \ldots, B_n \in ground_L(P)$ we denote a ground instance of a clause from P, and by $bag(a_1, \ldots, a_n)$ the multiset consisting of elements $a_1, \ldots, a_n$. Given a Herbrand interpretation I and a query Q we write $I \models Q$ if I is a model of Q. In particular, if A is a ground atom then $I \models A$ iff $A \in I$. Finally, T_P is the classical immediate consequence operator defined as follows: $T_P(I) = \{ A \in B_L \mid \exists A \leftarrow B_1, \ldots, B_n \in ground_L(P) \ : \ B_1, \ldots, B_n \in I \}$.

Specifications and Semantics

Following a Hoare's logic style of defining partial and total correctness, we stipulate that a *specification* is a pair $Pre, Post$ of (Herbrand) interpretations, i.e., subsets of B_L.

The rationale under this choice is the following. The first interpretation, Pre, specifies the *intended*, or *well-typed* one-atom queries, i.e., those queries for which we designed the program under consideration. The second interpretation, $Post$, specifies some desired property of successful one-atom queries. In this sense, a specification $(Pre, Post)$ describes the input-output behavior of a logic program, in a way that closely resembles that in Hoare's logic. According to this choice, the *well-typed* fragment of the least Herbrand model is $M_P \cap Pre$.

We are now ready to define our notions of (weak) partial and (weak) total correctness.

DEFINITION 1.1 Let P be a logic program, and $Pre, Post$ a specification.

- P is *partially correct* w.r.t. a specification $(Pre, Post)$ iff $M_P \cap Pre = Post$.
- P is *totally correct* w.r.t. a specification $(Pre, Post)$ iff $M_P \cap Pre = Post$ and $Pre \subseteq M_P \cup FF_P$, where FF_P is the finite failure set of P.

In addition, P is *weakly* partially or totally correct if the weaker requirement $M_P \cap Pre \subseteq Post$ holds. □

As a consequence of this definition, partial (and total) correctness of a program P w.r.t. a specification $(Pre, Post)$ entails that $Post$ coincides with the well-typed fragment of M_P.
As an example, the `APPEND` program is intuitively totally correct w.r.t. $Pre, Post$ where

$$\begin{aligned} Pre &= \{ \texttt{append}(xs,\ ys,\ zs) \mid xs, ys \text{ are lists } \} \\ Post &= \{ \texttt{append}(xs,\ ys,\ zs) \mid zs \text{ is the concatenation of the lists } xs \text{ and } ys \}. \end{aligned}$$

In the next section we introduce a proof theory for partial (and total) correctness which enables us to formalize the reasoning adopted in this example. In particular, our method allows us to abstract away from the ill-typed fragment of M_P.

Notice that the *weak* version of either notions of correctness entails that $Post$ specifies some property of $M_P \cap Pre$. For instance, the `APPEND` program is weakly totally correct w.r.t. $Pre, Post'$, where Pre is as above, and

$$Post' = \{ \texttt{append}(xs,\ ys,\ zs) \mid |zs| = |xs| + |ys| \}$$

where $|.|$ denotes the length of a list. Therefore, a postcondition in the sense of the weak correctness describes a property of every computed instance of an intended query.

Moreover, observe that the condition $Pre \subseteq M_P \cup FF_P$, used to define (weak) total correctness, is equivalent to requiring *existential termination* of the atoms in the precondition w.r.t. a fair selection rule and a complete search strategy. We will actually deal with *universal termination* with respect to the leftmost selection rule, namely Prolog's. This fact is immediately reflected by the definition of our proof method, in which a left-to-right ordering of clause bodies is taken into account. For this reason, the proposed proof theory for total correctness is not complete in this general sense, albeit it is complete for the class of so-called *left-terminating* programs, which, according to [AP94], includes most natural Prolog programs.

2 Proof Theory

We now introduce a proof method for the various notions of correctness, by means of the concept of (Hoare's logic style) triples $\{Pre\}\ P\ \{Post\}$ (for programs P) and $\{Pre\}\ Q\ \{Post\}$ (for queries Q). Triples are the basic tools to prove correctness. The proof method provides two distinct relations $\vdash$ and $\vdash_t$ on triples. We shall show how the relation $\vdash$ allows us to reason about (weak) partial correctness, whereas the relation $\vdash_t$ allows us to reason about (weak) total correctness.

2.1 The proof method

The next crucial definition explains the proof obligations needed to prove that the relations $\vdash$ or $\vdash_t$ hold on a given triple. As a consequence of the fact that a specification $(Pre, Post)$ of a program is assigned in terms of sets of ground atoms, we reason directly about ground instances of program clauses and queries. We recall from [Bez89] the notion of a *level mapping* (on a language L) which is a function $|\ |: B_L \rightarrow N$ from ground atoms to natural numbers. Level mappings play the role of termination functions in the proofs of total correctness.

Definition 2.1 Consider a program P and a query Q. Given a specification $(Pre, Post)$ we write:

- $\vdash_t \{Pre\}\ P\ \{Post\}$ iff there exists a level mapping $|\ |$ such that for every $A \leftarrow B_1, \ldots, B_n \in ground_L(P)$

  1. for $i \in [1, n]$
  $Pre \models A \quad \wedge \quad Post \models B_1, \ldots, B_{i-1} \Rightarrow$
  (a) $Pre \models B_i$ and
  (b) $|A| > |B_i|$

  2. $Pre \models A \quad \wedge \quad Post \models B_1, \ldots, B_n \Rightarrow Post \models A$

  We write $\vdash \{Pre\}\ P\ \{Post\}$ when *(1a)* and *(2)* hold. Pre is called a precondition and $Post$ a postcondition.

- $\vdash \{Pre\}\ Q\ \{Post\}$ iff for every ground instance $A_1, \ldots, A_n$ of Q

  3. for $i \in [1, n]$ $\quad Post \models A_1, \ldots, A_{i-1} \Rightarrow Pre \models A_i$

- $\vdash_t \{Pre\}\ Q\ \{Post\}$ iff there exist a level mapping $|\ |$ and $k \in N$ such that for every ground instance $A_1, \ldots, A_n$ of Q

  4. for $i \in [1, n]$ $\quad Post \models A_1, \ldots, A_{i-1} \Rightarrow Pre \models A_i \quad \wedge \quad k > |A_i|$ □

Intuitively, for a clause C with a body of length n, there are $n+1$ proof obligations to conclude that the relation $\vdash \{Pre\}\ P\ \{Post\}$ holds:

1. each atom B in the body of C is in Pre when the the head of C is in Pre and all the atoms to the left of B in the body of C are in $Post$;
2. the head of C is in $Post$ when it is in Pre and all the atoms in the body of C are in $Post$.

In the case of $\vdash_t \{Pre\}\ P\ \{Post\}$ the decreasing of the level mapping is also required. The left-to-right propagation of assumptions in proof obligations are biased by the left-to-right evaluation strategy of Prolog, in the sense that we require that a body atom is ready to be executed (i.e., it is in Pre) when the atoms to its left have been already executed (i.e., they are in $Post$). However, the presented notion is purely declarative, and no procedural intuition is needed to carry on the proof obligations.

Similar considerations apply to the notion of a triple for a query. We shall see in the sequel that the notion $\vdash_t \{Pre\}\ Q\ \{Post\}$ is a tool for proving that the query Q universally terminates.

The relation $\vdash$ is devised for proving (weak) partial correctness, whereas relation $\vdash_t$ allows us to prove (weak) total correctness: this role of triples will be clarified later. Some considerations about the above definition are in order.

Proving that a triple is in the relation $\vdash$ or $\vdash_t$ for a given program or query involves reasoning on their ground instances only. Basically, the definition provides a standard way for *lifting up* the results to non-ground queries. The advantage is that this lifting is made *a posteriori*.

Intuitively speaking, a precondition characterizes the intended (or well-typed) queries. A ground instance of a clause whose head is not in the precondition is superfluous as we never use it in a ground derivation starting with an atom of the precondition. The level mapping plays the role of a termination function. Strictly speaking, the level mapping has to be defined only on the precondition.

There are several symmetries in the Definitions 2.1. In particular, we want to point out that the Definitions 2.1 *(3,4)* for a query Q derive from Definitions 2.1 *(1,2)* for the program $goal \leftarrow Q$ where $goal$ is a new predicate. This follows from the following useful relation, which is immediate from Definition 2.1. For $\vdash \{Pre\}\ P\ \{Post\}$ and $A \leftarrow B_1, \ldots, B_n \in ground_L(P)$,

$$A \in Pre \quad \text{implies} \quad \vdash \{Pre\}\ B_1, \ldots, B_n\ \{Post\} \tag{1}$$

Proof Outlines

The proof that $\vdash_t \{Pre\}\ P\ \{Post\}$ holds can be presented in a suggestive way by means of *proof outlines*, a well-known tool in verification of imperative programs.

DEFINITION 2.2 A proof outline for a clause $A \leftarrow A_1, \ldots, A_n$ and $|\ |$, Pre, $Post$ is a labelled clause of the form

$$\begin{array}{llll}
\{g_0\} & & & \\
A_0 & \leftarrow & & \{t_0\} \\
 & & \{g_1\} & \\
 & A_1, & & \{t_1\} \\
 & & \{f_1\} & \\
 & & \{g_2\} & \\
 & \vdots & & \\
 & A_{n-1}, & & \{t_{n-1}\} \\
 & & \{f_{n-1}\} & \\
 & & \{g_n\} & \\
 & A_n. & & \{t_n\} \\
 & & \{f_n\} & \\
\{f_0\} & & &
\end{array}$$

where t_i and f_i, g_i, for $i \in [0, n]$ are respectively integer expressions and assertions (in some formal logic), such that every ground instance of the following *proof obligations* is satisfied:

(i) for $i \in [0, n]$: $t_i = |A_i|$,

(ii) for $i \in [0, n]$: $g_i \Leftrightarrow A_i \in \mathit{Pre}$,

(iii) for $i \in [0, n]$: $f_i \Leftrightarrow A_i \in \mathit{Post}$,

(iv) for $i \in [1, n]$: $g_0 \wedge f_1 \wedge \ldots \wedge f_{i-1} \Rightarrow g_i \wedge t_0 > t_i$,

(v) $g_0 \wedge f_1 \wedge \ldots \wedge f_n \Rightarrow f_0$. □

The intuition is that the assertion g_i specifies the conditions under which the atom A_i of the clause is in *Pre*, the assertion f_i specifies the conditions under which A_i is in *Post*, and the expressions t_i represent the level mapping of A_i. Under this assumption, the proof obligations (iv) and (v) directly reflect respectively Definition 2.1 *(1)* and *(2)*.

By construction, we have that $\vdash_t \{\mathit{Pre}\}\ P\ \{\mathit{Post}\}$ if and only if there exists a level mapping $|\ |$ and a proof outline for each clause of P and $|\ |, \mathit{Pre}, \mathit{Post}$.

Consider, as an example, the following proof outline for the recursive clause of `APPEND`.

$$
\begin{array}{ll}
\{\ \texttt{[}X\,\texttt{|}\,Xs\texttt{]} \text{ and } Ys \text{ lists }\} & \\
\texttt{append([X|Xs], Ys, [X|Zs])} \leftarrow & \{\ |Xs|+1\ \} \\
\quad \{\ Xs \text{ and } Ys \text{ lists }\} & \\
\quad \texttt{append(Xs, Ys, Zs).} & \{\ |Xs|\ \} \\
\quad \{\ Zs \text{ is the concatenation of } Xs \text{ and } Ys\ \} & \\
\{\ \texttt{[}X\,\texttt{|}\,Zs\texttt{]} \text{ is the concatenation of } \texttt{[}X\,\texttt{|}\,Xs\texttt{]} \text{ and } Ys\ \} &
\end{array}
$$

where *Pre* and *Post* are chosen as in Section 1, and the level mapping for an `append` atom is chosen as the length $|Xs|$ of its first argument Xs. It is immediate to verify that the proof obligations are satisfied.

Proof outlines become simpler when considering the relation $\vdash$. In the labeled clause no level mapping t_i appears and the proof obligations are (ii-iii-v) plus the following simplified version of (iv):

(iv$'$) for $i \in [1, n]$: $g_0 \wedge f_1 \wedge \ldots \wedge f_{i-1} \Rightarrow g_i$.

Again, by construction $\vdash \{\mathit{Pre}\}\ P\ \{\mathit{Post}\}$ holds iff there exists a proof outline for each clause of P and $\mathit{Pre}, \mathit{Post}$.

It should be observed how proof outlines represent in a concise way the domino-style propagation of premises in the proof obligations of Definition 2.1.

In the sequel, we shall also make use of a weakened form of proof outlines, which enable us to use assertions f_i and g_i which do not coincide with post and pre-conditions, but rather represent strengthenings and weakenings of *Post* and *Pre*, according to the following equivalent interpretation of the assertions used in proof outlines. We denote with h_i for $i \in [1, n+1]$ the assertion $g_0 \wedge f_1 \wedge \ldots \wedge f_{i-1}$.

(i) for $i \in [0, n]$: $h_i \Rightarrow t_i = |A_i|$,

(ii) $g_0 \Leftarrow A_0 \in \mathit{Pre}$, and for $i \in [1, n]$: $h_i \wedge g_i \Rightarrow A_i \in \mathit{Pre}$,

(iii) for $i \in [1, n]$: $f_i \Leftarrow A_i \in \mathit{Post} \wedge h_i$, and $h_n \wedge f_0 \Rightarrow A_0 \in \mathit{Post}$,

(iv) for $i \in [1, n]$: $h_i \Rightarrow g_i \wedge t_0 > t_i$.

(v) $h_n \Rightarrow f_0$,

With this more liberal interpretation, simpler proof outlines can be constructed. The construction of a proof outline satisfying the proof obligations (i) through (v) for every clause in a program P amounts to showing that $\vdash_t \{\mathit{Pre}\}\ P\ \{\mathit{Post}\}$ holds. In the following figure it is depicted how (i) through (iv) imply conditions *1(a)* and *1(b)* of Definition 2.1.

$$\begin{array}{lllllll}
g_0 & \wedge\ f_1 & \wedge\ \ldots & \wedge & f_{i-1} & \Rightarrow\ g_i & \wedge\ t_0 > t_i \\
\Uparrow & \quad\Uparrow & & & \Uparrow & \quad\Downarrow & \quad\Downarrow \\
A_0 \in \mathit{Pre} & \wedge\ A_1 \in \mathit{Post} & \wedge\ \ldots & \wedge & A_{i-1} \in \mathit{Post} & \Rightarrow\ A_i \in \mathit{Pre} & \wedge\ |A_0| > |A_i|
\end{array}$$

Similarly, the proof obligations (i,ii,iii,v) imply condition *2* of Definition 2.1.

3 Partial Correctness

3.1 Weak Partial Correctness

In this section, we recall some properties of the relation $\vdash$ (see, e.g., [PR96] or [AM94]) and apply them to our case study. First, a justification of the intuitive notion that the postcondition is a description of the correct instances of the queries which satisfy the relation $\vdash \{\mathit{Pre}\}\ Q\ \{\mathit{Post}\}$ is provided.

LEMMA 3.1 *Let P be a program and Q a query such that $\vdash \{\mathit{Pre}\}\ P\ \{\mathit{Post}\}$ and $\vdash \{\mathit{Pre}\}\ Q\ \{\mathit{Post}\}$. Then:*
(i) for every SLD-resolvent Q' of Q and P the relation $\vdash \{\mathit{Pre}\}\ Q'\ \{\mathit{Post}\}$ holds, and
(ii) for every computed (or correct) instance Q' of Q and P we have $\mathit{Post} \models Q'$. □

As a direct consequence, the relation $\vdash$ is a proof method for weak partial correctness, i.e. that $\vdash \{\mathit{Pre}\}\ P\ \{\mathit{Post}\}$ implies $M_P \cap \mathit{Pre} \subseteq \mathit{Post}$. we will provide later a strengthening of the method to prove the other inclusion, namely partial correctness.

THEOREM 3.2 (WEAK PARTIAL CORRECTNESS) *A program P such that $\vdash \{\mathit{Pre}\}\ P\ \{\mathit{Post}\}$ is weak partially correct w.r.t. the specification $(\mathit{Pre}, \mathit{Post})$.*

Proof. Consider $A \in M_P \cap \mathit{Pre}$. Then $\vdash \{\mathit{Pre}\}\ A\ \{\mathit{Post}\}$ holds, and, by Completeness of SLD-resolution, the query A has an SLD-refutation. So $A \in \mathit{Post}$ by Lemma 3.1 *(ii)*. □

3.2 Partial Correctness

The intuition underlying an *Hoare style* proof method based on *Pre*/*Post*-conditions becomes more concrete in the present context when dealing with modular proofs. The following modularity rule reflects an obvious property of triples:

$$\frac{\vdash \{\mathit{Pre}\}\ P\ \{\mathit{Post}\} \quad \vdash \{\mathit{Pre}\}\ P\ \{\mathit{Post}'\}}{\vdash \{\mathit{Pre}\}\ P\ \{\mathit{Post} \cap \mathit{Post}'\}}.$$

The importance of this fact is twofold. On the one hand, it allows us to split a correctness proof into simpler ones.
On the other hand, it allows us to introduce a notion of *strongest postcondition.*

DEFINITION 3.3 Let P be a program such that $\vdash \{\mathit{Pre}\}\ P\ \{\mathit{Post}\}$. By $sp(P, \mathit{Pre})$ we denote the intersection of all Post' such that $\vdash \{\mathit{Pre}\}\ P\ \{\mathit{Post}'\}$. □

This definition, however, is not very useful to construct $sp(P, \mathit{Pre})$. We provide an alternative characterization, by relating $sp(P, \mathit{Pre})$ to $M_P \cap \mathit{Pre}$. As $sp(P, \mathit{Pre})$ is an intersection of postconditions, each of which includes $M_P \cap \mathit{Pre}$ by Theorem 3.2, we have the inclusion $sp(P, \mathit{Pre}) \supseteq M_P \cap \mathit{Pre}$. The reverse inclusion can be proved by observing that the interpretation $M_P \cap \mathit{Pre}$ is a postcondition if $\vdash \{\mathit{Pre}\}\ P\ \{\mathit{Post}\}$ holds. This fact is an immediate consequence of the inclusion $M_P \cap \mathit{Pre} \subseteq \mathit{Post}$ implied by Theorem 3.2. Thus, $M_P \cap \mathit{Pre}$ is indeed the strongest postcondition as recorded in the following result.

THEOREM 3.4 *For a program P such that $\vdash \{Pre\}\ P\ \{Post\}$ we have $\vdash \{Pre\}\ P\ \{M_P \cap Pre\}$. Hence, $sp(P, Pre) = M_P \cap Pre$.* □

As a result we have the following Partial Correctness Theorem.

THEOREM 3.5 (PARTIAL CORRECTNESS) *A program P such that $\vdash \{Pre\}\ P\ \{Post\}$ is partially correct w.r.t. the specification $(Pre, sp(P, Pre))$.* □

The problem is how to characterize the strongest postcondition without first constructing the least Herbrand model M_P. To this end, we introduce the notion of a *well-supported interpretation*, which are well-studied in the context of semantics for general logic programs (see [AB94]).

DEFINITION 3.6 Suppose that $\vdash \{Pre\}\ P\ \{Post\}$.
Post is a *well-supported interpretation* (w.r.t. P and *Pre*) iff there exist

- a well-founded poset $(W, <)$,
- and a function $|\ | : B_L \rightarrow W$

such that for any $A \in Post \cap Pre$ there exists $A \leftarrow B_1, \ldots, B_n \in ground_L(P)$ such that

$$\forall i \in [1, n] : Post \models B_i \wedge |A| > |B_i|.$$

□

The idea underlying this definition is to require that any atom in $Post \cap Pre$ has a successful ground derivation. In fact, for any such atom there exists a ground derivation which is finite, because the poset is well-founded, and successful, because the last selected atom unifies with some clause head. Therefore, a well-supported post-condition is a subset of the interesting fragment of the least Herbrand model. Therefore, well-supportedness combined with the requirements of Definition 2.1 entails that indeed M_P coincides with a well-supported interpretation on the atoms in *Pre*. Observe that, in Definition 3.6, the condition $Post \cap Pre \models B_i$ for $i \in [1, n]$ is equivalent to $Post \models B_i$ for $i \in [1, n]$ under the assumption that $\vdash \{Pre\}\ P\ \{Post\}$ holds.
The next result states that well-supported post-conditions provide a proof method for partial correctness, which is complete for programs satisfying the relation $\vdash$. A proof of this result can be found in [PR96].

THEOREM 3.7 *Let P be a program such that $\vdash \{Pre\}\ P\ \{Post\}$. Then*

$$Post \cap Pre = sp(P, Pre) \quad \textit{iff} \quad Post \ \textit{ is well-supported (w.r.t. } P \textit{ and } Pre\textit{).}$$

□

Observe that if $\vdash \{Pre\}\ P\ \{Post\}$ holds then by Theorem 3.4 $\vdash \{Pre\}\ P\ \{sp(P, Pre)\}$ holds and $sp(P, Pre) \cap Pre = sp(P, Pre)$ which implies that $sp(P, Pre)$ is well-supported by the Theorem above.

Proof Outlines

Proving weak partial correctness is handy, as the task can be carried out using the proof outlines. In contrast, Definition 3.6 may seem intricate and difficult to handle with simple tools. Fortunately, it has a straightforward interpretation in terms of proof outlines.

Definition 3.8 A proof outline for a clause $A \leftarrow A_1 , \ldots , A_n$, a function $|\ | : B_L \rightarrow W$ into a well-founded poset $(W, <)$ and *Pre*, *Post*, is a labelled clause of the form

$$
\begin{array}{llll}
\{g\} & & & \\
A_0 & \leftarrow & & \{t_0\} \\
 & & A_1, & \{t_1\} \\
 & & \quad \{f_1\} & \\
 & & \vdots & \\
 & & A_{n-1}, & \{t_{n-1}\} \\
 & & \quad \{f_{n-1}\} & \\
 & & A_n. & \{t_n\} \\
 & & \quad \{f_n\} &
\end{array}
$$

where t_i for $i \in [0, n]$ and f_i, g, for $i \in [1, n]$ are respectively integer expressions and assertions (in some formal logic), such that every ground instance of the following *proof obligations* holds:

(i) for $i \in [0, n]$: $g \Rightarrow t_i = |A_i|$,

(ii) for $i \in [1, n]$: $g \wedge f_i \Rightarrow A_i \in Post$,

(iii) for $i \in [1, n]$: $g \Rightarrow f_i \wedge t_0 > t_i$. □

The assertion g expresses a relation among the variables of the clause in such a way that for every ground instance of the proof outline if the instance of g holds then $A_i \in Post \wedge |A_0| > |A_i|$. The use of the auxiliary assertions f_i's is not strictly necessary, albeit useful in constructing proof outlines.

Next, in order to prove that *Post* is well-supported w.r.t. P and *Pre* we have to show that there exist a set of proof outlines for (instances of) clauses from P and a same function $|\ | : B_L \rightarrow W$ such that for every atom A in $Post \cap Pre$ is a ground instance of some head atom in a proof outline and the assertion associated with the head holds.

4 Total Correctness

The relation $\vdash$ is not sufficient for dealing with (weak) total correctness, as termination is not taken into account. Relation $\vdash_t$ deals with termination by means of level mappings, which decrease from the head to the body atoms of a ground instance of a clause (see Definition 2.1).
In addition to the properties of $\vdash$, the relation $\vdash_t$ allows to prove *universal termination* w.r.t. *LD* resolution, i.e. the resolution strategy adopted in Prolog. The next result shows that, when $\vdash_t \{Pre\}\ P\ \{Post\}$ and $\vdash_t \{Pre\}\ Q\ \{Post\}$ any *LD*-tree for $P \cup \{Q\}$ is finite, and vice versa [Rug94, PR96].

Theorem 4.1 *If* $\vdash_t \{Pre\}\ P\ \{Post\}$ *holds then*

$$\text{for any } A \in Pre \text{ every } LD\text{-tree for } P \text{ and } A \text{ is finite.} \tag{2}$$

Conversely, $\vdash \{Pre\}\ P\ \{Post\}$ *and (2) imply* $\vdash_t \{Pre\}\ P\ \{sp(P, Pre)\}$. □

As an immediate corollary of this termination property, we have the Total Correctness Theorems.

Theorem 4.2 ((Weak) Total Correctness) *A program P such that* $\vdash_t \{Pre\}\ P\ \{Post\}$ *is weak totally correct w.r.t. the specification* $(Pre, Post)$*, and totally correct w.r.t. the specification* $(Pre, sp(P, Pre))$. □

The next step is to find a proof method for total correctness. A straightforward consequence of the results of Section 3.2 serves our purposes.

Theorem 4.3 *Let P be a program such that $\vdash_t \{Pre\}\ P\ \{Post\}$. Then the following statements are equivalent:*

(i) Post is well-supported w.r.t. P and Pre,

(ii) $Post \cap Pre = sp(P, Pre)$,

(iii) $T_P(Post) \supseteq Post \cap Pre$.

□

5 Relation $semi\!-\vdash_t$

The definition of the relation $\vdash_t$ is not fully satisfactory with respect to the expressiveness of the level mapping used. Let us see this problem with an example. Consider the program SUBLIST

```
sublist(Xs, Ys)  ← Xs is a sublist of Ys

sublist(Xs, Ys) ←
   append(_, Zs, Ys),
   append(Xs, _, Zs).
```

joinly to APPEND.

A level mapping such that

$$|\mathtt{append}(\_,\_,[x_1,\ldots,x_n])| = n$$

seems a natural candidate to show $\vdash_t \{Pre_{\mathtt{APPEND}}\}$ APPEND $\{Post_{\mathtt{APPEND}}\}$, where

$$Pre_{\mathtt{APPEND}} = \{\mathtt{append}(\_,\_,xs) \mid xs \in List(\alpha)\}$$
$$Post_{\mathtt{APPEND}} = \{\mathtt{append}(xs,\ ys,\ zs) \mid zs = xs * ys\}$$

Analogously, considering the *pre*-condition for SUBLIST

$$Pre_{\mathtt{SL}} = \{\mathtt{sublist}(\_,xs) \mid xs \in List(\alpha)\} \cup Pre_{\mathtt{APPEND}}$$

an intuitive candidate level mapping is $|\mathtt{sublist}(\_,[x_1,\ldots,x_n])| = n$. However, the proof obligations on the decreasing of the level mapping require

$$|\mathtt{sublist}(\_,[x_1,\ldots,x_n])| > |\mathtt{append}([x_1,\ldots,x_n],\_,\_)| = n.$$

Therefore, to show that $\vdash_t \{Pre_{\mathtt{APPEND}}\}$ APPEND $\{Post_{\mathtt{APPEND}}\}$ holds we must consider a somewhat unnatural level mapping, i.e. $|\mathtt{sublist}(\_,[x_1,\ldots,x_n])| = n{+}1$. We point out that such a phenomenon propagates upward when considering programs which use SUBLIST. However, the counter-intuitiveness of the level mapping is not the only drawback of relation $\vdash_t$. To some extend, one can assume that the implementation of APPEND is hidden, and only its specification is known which consists of *pre/post-conditions* + level mapping. Suppose that the implementation of APPEND is modified later on to the following

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

list([]).
list([X|Xs]) ←
   list(Xs).
```

For the same motivation of SUBLIST, the level mapping for APPEND must be set as follows

$$|\texttt{append}(\_,\_,[x_1,\dots,x_n])| > n.$$

This fact modifies the specification of APPEND. All the modules that use APPEND have to be reconsidered, and as a result this problem propagates upward. In the following we provide a relation *semi*$-\vdash_t$ that is a reformulation of $\vdash_t$ for which this modularity problem does not arise.

Stratification

First, we introduce some notations.

DEFINITION 5.1 Let P be a program and p, q predicate symbols.

(i) We say that p *refers to* q *in* P iff there is a clause from P that use p in the head and q in the body.

(ii) We say that p *depends on* q *in* P, and write $\sqsupseteq q$, iff (p, q) is in the reflexive transitive closure of the relation *refers to*.

(iii) We write $p \simeq q$ iff $p \sqsupseteq q$ and $q \sqsupseteq p$.

(iv) We write $p \sqsupset q$ iff $p \sqsupseteq q$ and $q \not\sqsupseteq p$. □

In other words $p \simeq q$ means that p and q are mutually recursive predicates, whereas $p \sqsupset q$ means that q is used in the implementation of p, but not vice versa. Let us extend the relations to programs.

DEFINITION 5.2

- Given a program P and a predicate symbol p, the *definition of* p (w.r.t. P) is the set of clauses from P whose head has the predicate symbol p.
- A partition $P = P_1 \uplus \dots \uplus P_n$ is called a *stratification* for P iff the definition of every relation that appears in a clause from P_i is contained in
  $$\bigcup_{j \le i} P_i.$$
  Each P_i is called a *stratum*.
- A relation p is *defined on* a program P iff p occurs in the head of a clause from P. Given two programs P and Q, we say that P *extend* Q if no relation defined in P occurs in Q. □

The key notion of stratification expresses a way to decompose a program in a modular way. The following lemma taken from [Apt90] shows the relation between stratification and the extensions of sub-programs. An immediate consequence of the lemma is that $p \simeq q$ implies p, q are in the same stratum.

LEMMA 5.3 *A partition* $P_1 \uplus \dots \uplus P_n$ *of a program* P *is a stratification iff, for every* $i = 1, \dots n$, P_i *extends*

$$\bigcup_{j < i} P_i.$$

□

Relation $semi\!-\vdash_t$

We provide a reformulation of $\vdash_t$ which takes into account the modular organization of the program under consideration.

Definition 5.4 Given a program P and a specification $(Pre, Post)$ we write $\vdash_t \{Pre\}\ P\ \{Post\}$ iff there exists a level mapping $|\ |$ such that for every $A \leftarrow B_1, \ldots, B_n \in ground_L(P)$

1. for $i \in [1, n]$:

$$Pre \models A \quad \wedge \quad Post \models B_1, \ldots, B_{i-1} \Rightarrow$$

$$\begin{array}{ll} (a) & Pre \models B_i \quad \text{and} \\ (b) & |A| > |B_i| \ \text{if}\ rel(A) \simeq rel(B) \\ & |A| \geq |B_i| \ \text{if}\ rel(A) \sqsupset rel(B) \end{array}$$

2. $Pre \models A \quad \wedge \quad Post \models B_1, \ldots, B_n \Rightarrow Post \models A$

□

It is not necessary to define a relation $semi\!-\vdash_t$ for queries. We point out that $\vdash_t \{Pre\}\ P\ \{Post\}$ implies $semi\!-\vdash_t \{Pre\}\ P\ \{Post\}$. Indeed, the two relations coincide, as the following theorem shows.

Theorem 5.5 *For every P and $Pre, Post$: $\vdash_t \{Pre\}\ P\ \{Post\}$ iff $semi\!-\vdash_t \{Pre\}\ P\ \{Post\}$.*

Proof. The only-if part is trivial. Consider the if-part. We have only to show the decreasing of the level mapping. In Apt [Apt90] it is shown that every program is stratifiable. Consider then a stratification $\Pi_1, \ldots, \Pi_n$ of P. Assuming $|p|' = i$ iff $p \in \Pi_i$, we define the level mapping

$$\|A\| = |A| + |rel(A)|'.$$

and show that $\vdash_t \{Pre\}\ P\ \{Post\}$ holds by $\|\ \|$. Let $A \leftarrow B_1, \ldots, B_n$ be a ground instance of a clause from P, $i \in [1, n]$ and

$$Pre \models A \ \wedge \ Post \models B_1, \ldots, B_{i-1}.$$

We distinguish two cases. If $rel(A) \simeq rel(B_i)$ then they are in the same Π_k for some k. Therefore

$$\begin{array}{lll} \|A\| & = & |A| + |rel(A)|' \\ & = & \quad \{ \text{ Definition of } |\ |' \ \} \\ & & |A| + |rel(B_i)|' \\ & > & \quad \{ \text{ Definition 5.4 } \} \\ & & |B_i| + |rel(B_i)|' = \|B_i\|. \end{array}$$

If $rel(A) \sqsupset rel(B_i)$ then $rel(A) \in \Pi_k$ and $rel(B_i) \in \Pi_j$ with $k > j$. Since $|rel(A)|' > |rel(B_i)|'$ we have

$$\begin{array}{lll} \|A\| & = & |A| + |rel(A)|' \\ & > & \quad \{ \text{ Definition of } |\ |' \ \} \\ & & |A| + |rel(B_i)|' \\ & \geq & \quad \{ \text{ Definition 5.4 } \} \\ & & |B_i| + |rel(B_i)|' = \|B_i\|. \end{array}$$

□

This result allows us to extend all the properties of $\vdash_t$ to $semi\!-\vdash_t$.

Proof Outlines

Proof outlines for *semi*$-\vdash_t$ are a small variant of those for $\vdash_t$. The only change is in the proof obligation *(v)*, that becomes

(v') for $i \in [1, n]$

$$g'_0 \wedge f'_1 \wedge \ldots \wedge f'_{i-1} \Rightarrow g'_i \wedge \begin{cases} t'_0 > t'_i \text{ if } rel(A_0) \simeq rel(A_i) \\ t'_0 \geq t'_i \text{ if } rel(A_0) \sqsupset rel(A_i) \end{cases}$$

Consider `SUBLIST` again. By defining

$$|\texttt{append}(\_, \_, [x_1, \ldots, x_n])| = n$$
$$|\texttt{sublist}(\_, [x_1, \ldots, x_n])| = n$$

it results that *semi*$-\vdash_t$ $\{Pre_{\texttt{SL}}\}$ `SUBLIST` $\{Post_{\texttt{SL}}\}$ where

$$Post_{\texttt{SL}} = \{\texttt{sublist}(xs,\ ys) \mid xs \text{ is a sublist of } ys\} \cup Post_{\texttt{APPEND}}$$

The proof outline for the first clause is

$\{\ ys \in List(\alpha)\ \}$
`sublist(Xs, Ys)` $\leftarrow$ $\qquad\{\ |Ys|\ \}$
$\quad\{\ ys \in List(\alpha)\ \}$
$\quad$`append(_, Zs, Ys)` $\qquad\{\ |Ys|\ \}$
$\quad\{\ ys = \_ * zs\ \}$
$\quad\{\ zs \in List(\alpha)\ \}$
$\quad$`append(Xs, _, Zs)` $\qquad\{\ |Zs|\ \}$
$\quad\{\ zs = xs * \_\ \}$
$\{\ ys = \_ * xs * \_\ \}$

We point out that changing the implementation of `APPEND` does not involve to reconsider the correctness proofs of the modules using `APPEND`, since the level mapping for it does not change.

5.1 Modular Proofs

The real applicability of a proof method is subject to the availability of tools that help in the development of large projects. Logic programming is a natural paradigm for declarative programming. In addition, it allows to reason modularly by the simple observation that a piece of a program is still a program. The concept of stratification is fundamental in this context. An program P is naturally structured by means of a stratification $\pi_1, \ldots, \pi_n$. The analysis of P may be performed incrementally by considering for $i = 1, 2 \ldots n$ the set of clauses in the definition of the predicates of

$$\bigcup_{k=1}^{i} \pi_k.$$

In the following we will give some tools to help the modular verification of a program. Fist, we make this natural assumption.

> **Assumption.** *The theorems of this section implicitely require that in the pre/post-conditions taken under consideration for the analysis of a program P only predicate symbols from Π_P appear.*

This assumption is not a restriction since the following fact hold when $\vdash \{Pre\}\ P\ \{Post\}$

- $\vdash \{Pre|_{\Pi_P}\}\ P\ \{Post|_{\Pi_P}\}$, and
- $\vdash \{Pre \cup I\}\ P\ \{Post \cup J\}$ for every $I, J \subseteq B_{\langle \Sigma_L, \Pi_L \setminus \Pi_P \rangle}$, and
- $sp(P, Pre)|_{\Pi_P} = sp(P, Pre) = sp(P, Pre|_{\Pi_P})$.

Relation $\vdash$

Our objective is to show that a triple for $P \cup Q$ holds starting from triples for P and Q. Assuming that we have proven a triple for the clauses of a program in the definition of $\cup_{k=1}^{i-1}\pi_k$, the expected result allows us to incrementally show that a triple holds for the clauses of the program in the definition of $\pi_i \cup \cup_{k=1}^{i-1} \pi_k = \cup_{k=1}^{i}\pi_k$. Iterating the process for $i = 1, 2..n$ one ends up with a triple for the whole program with the advantage of having split the problem into several simpler ones.

THEOREM 5.6 *Let P, Q be two programs such that P extends Q. Suppose that:*
(i) $\vdash \{Pre_P\}\ P\ \{Post_P\}$ *and* $\vdash \{Pre_Q\}\ Q\ \{Post_Q\}$
(ii) $Pre_P|_{\Pi_Q} \subseteq Pre_Q|_{\Pi_P}$
(iii) $Post_P|_{\Pi_Q} = Post_Q|_{\Pi_P}$

Then the triple $\vdash\ \{Pre_P \cup Pre_Q\}\ \ P \cup Q\ \ \{Post_P \cup Post_Q\}$ *holds. Moreover, if $Post_Q$ is a well-supported interpretation (w.r.t. Q and Pre_Q) and there exist a well-founded irreflexive partial order $(W_P, <_P)$ and a function $|\ |_P : B_L \rightarrow W$ such that*

$$\forall A \in Post_P|_{\Pi_P \setminus \Pi_Q} \cap Pre_P\ \exists A \leftarrow B_1, \ldots, B_n \in ground_L(P)\ \ \forall i \in [1,n] : Post \models B_i \wedge |A|_P >_P |B_i|_P$$

then also $Post_P \cup Post_Q$ is a well-supported interpretation (w.r.t. $P \cup Q$ and $Pre_P \cup Pre_Q$.)

Proof. The proofs of this theorem and of the next two ones are reported in the Appendix A. □

A good example for explaining this result is the following. Consider Q as an *external module* (possibily written in a programming language different from Prolog), and $Pre_Q, Post_Q$ as its specification. The integration of the code is declaratively transparent to the programmer who only knows the specification of the external module.

Relation $\vdash_t$

As for $\vdash$, we propose a method to show in a modular way that a triple in the relation $\vdash_t$ holds.

THEOREM 5.7 *Let P, Q be two programs such that P extends Q. Suppose that:*
(i) $\vdash_t \{Pre_P\}\ P\ \{Post_P\}$ *holds by* $|\ |_P$,
(ii) $\vdash_t \{Pre_Q\}\ Q\ \{Post_Q\}$ *holds by* $|\ |_Q$,
(iii) $Pre_P|_{\Pi_Q} \subseteq Pre_Q$
(iv) $Post_P|_{\Pi_Q} = Post_Q|_{\Pi_P}$
(v) $\forall A \leftarrow B_1, \ldots, B_n \in ground_L(P) \quad \forall\ i \in [1,n]$ *if* $Pre_P \models A$ *and* $Post_P \models B_1, \ldots, B_n$ *then*

$$|\ A\ |_P\ >\ |\ B_i\ |_Q\ .$$

Then the triple $\vdash \{Pre_P \cup Pre_Q\}\ P \cup Q\ \{Post_P \cup Post_Q\}$ *holds by* $|\ |: B_L \rightarrow N$ *defined as follow:*

$$|\ A\ |\ = \begin{cases} |\ A\ |_P & \textit{if rel(A) is defined in P,} \\ |\ A\ |_Q & \textit{if rel(A) is defined in Q,} \\ 0 & \textit{otherwise} \end{cases}$$

Moreover, if $Post_Q$ is well-supported w.r.t. Q and Pre_Q and

(vi) $T_P(Post_P) \supseteq Post_P|_{\Pi_P \setminus \Pi_Q} \cap Pre_P$
then $Post_P \cup Post_Q$ is well-supported w.r.t. $P \cup Q$ and $Pre_P \cup Pre_Q$. □

Relation $semi\!-\vdash_t$

Theorem 5.7 still makes too strong assumptions. In fact, the level mapping for P has to satisfy condition *(v)* which is dependent on the level mapping developed for Q. This is in contradiction with the two-step development process consisting of a separate analysis of P and Q first, and then the analysis of $P \cup Q$ then. The next result copes with this problem by providing rather general conditions for a modular proof of a triple in the relation $semi\!-\vdash_t$.

THEOREM 5.8 *Let P, Q be two programs such that P extends Q. Suppose that:*
(i) $semi\!-\vdash_t \{Pre_P\}\ P\ \{Post_P\}$ *holds by* $|\ |_P$,
(ii) $semi\!-\vdash_t \{Pre_Q\}\ Q\ \{Post_Q\}$ *holds by* $|\ |_Q$,
(iii) $Pre_P|_{\Pi_Q} \subseteq Pre_Q$
(iv) $Post_P|_{\Pi_Q} = Post_Q|_{\Pi_P}$
(v) there exists a function $\|\ \| : B_{\langle \Sigma_L, \Pi_P \rangle} \rightarrow Z$ *such that*

- $\forall A \leftarrow B_1, \ldots, B_n \in ground_L(P) \quad \forall\ i \in [1, n]$ *if* $Pre_P \models A$ *and* $Post_P \models \mathbf{A}$ *then*
  - *(a)* $\|A\| \geq \|B_i\|$ *if rel(B) is defined in P,*
  - *(b)* $\|A\| \geq |\ B_i\ |_Q\ -\ |\ A\ |_P$ *if rel(B) is defined in Q,*
- *for every* $A \in B_L$
  - *(c)* $|\ A\ |_P\ + \|A\| \geq 0$ *if rel(A) is defined in P.*

Then the triple $semi\!-\vdash \{Pre_P \cup Pre_Q\}\ P \cup Q\ \{Post_P \cup Post_Q\}$ *holds by* $|\ |: B_L \rightarrow N$ *defined as follow:*

$$|\ A\ | \ = \begin{cases} |\ A\ |_P\ + \|A\| & \textit{if rel(A) is defined in P,} \\ |\ A\ |_Q & \textit{if rel(A) is defined in Q,} \\ 0 & \textit{otherwise} \end{cases}$$

and $Pre_P \cup Pre_Q, Post_P \cup Post_Q$.

Moreover, if $Post_Q$ *is well-supported w.r.t.* Q *and* Pre_Q *and*
(vi) $T_P(Post_P) \supseteq Post_P|_{\Pi_P \setminus \Pi_Q} \cap Pre_P$
then $Post_P \cup Post_Q$ *is well-supported w.r.t.* $P \cup Q$ *and* $Pre_P \cup Pre_Q$. □

The function $\|\ \|$ expresses the *difference* to add to $|\ |_P$ to dominate $|\ |_Q$.

EXAMPLE 5.9 Consider the program `SELECTS = SEL ∪ SELS`

```
selects(Xs, Ys)  ← Xs ⊆ Ys

selects([],Ys).
selects([X | Xs], Ys) ←
   select(X, Ys, Ys1),
   selects(Xs, Ys1).

select(X, [X | Xs], Xs).
select(X, [Y | Xs], [Y| Ys]) ←
   select(X, Xs, Ys).
```

where `SEL` is the definition of `select` and `SELS` that of `selects`. Suppose to develop `SEL` and `SELS` independentely, with the following specifications
`SEL`

$$Pre_{\texttt{SEL}} = \texttt{select}(U_L \times List(\alpha) \times U_L)\ \cup\ \texttt{select}(U_L \times U_L \times List(\beta))$$
$$Post_{\texttt{SEL}} = \{\texttt{select}(x,\ xs,\ ys) \mid xs = [x_1, \ldots, x_n, x|t] \wedge ys = [x_1, \ldots, x_n|t]\}$$
$$|\texttt{select}(x,\ xs,\ ys)| = \begin{cases} min\{|xs|, |ys|\} & \text{if } xs \in List(\alpha) \wedge ys \in List(\beta) \\ |xs| & \text{if } xs \in List(\alpha) \wedge ys \notin List(\beta) \\ |ys| & \text{if } xs \notin List(\alpha) \wedge ys \in List(\beta) \end{cases}$$

SELS

$$Pre_{\texttt{SELS}} = \texttt{selects}(List(\alpha) \times List(\alpha)) \ \cup\ \texttt{select}(\alpha \times List(\alpha) \times List(\alpha))$$

$$Post_{\texttt{SELS}} = \{\texttt{select}(xs,\ ys) \mid xs \subseteq ys\} \ \cup\ Post_{\texttt{SEL}}$$

$$|\texttt{selects}(xs,\ ys)| = |xs|$$

$$|\texttt{select}(x,\ xs,\ ys)| = 0$$

The reader can easily check that $semi\!-\vdash_t \{Pre_{\texttt{SEL}}\}$ SEL $\{Post_{\texttt{SEL}}\}$ and $semi\!-\vdash_t \{Pre_{\texttt{SELS}}\}$ SELS $\{Post_{\texttt{SELS}}\}$ hold. We point out that the level mapping for SELS is not as one may exspect. If we consider the whole SELECTS, it should be

$$|\texttt{selects}(xs,\ ys)| = |ys|$$

where

$$Pre_{\texttt{SELECTS}} = \texttt{selects}(U_L \times List(\alpha)) \ \cup\ \texttt{select}(\alpha \times List(\alpha) \times List(\alpha))$$

This way, $\vdash_t \{Pre_{\texttt{SELS}}\}$ $\texttt{selects(X,}\ ys$ $\{Pre_{\texttt{SELS}}\}$ would hold. Let us show the hypothesis of Theorem 5.8. We point out that for $xs, ys \in List(\alpha)$

$$\texttt{select}(x,\ xs,\ ys) \in Post_{SEL} \quad \Rightarrow \quad |ys| = |xs| + 1$$

Therefore, we have to find a funcion $\|\ \|$ such that:

$$|ys| = |ys1| + 1 \quad \Rightarrow \begin{cases} \|\texttt{selects([}x\ \texttt{|}\ xs\texttt{]},\ ys)\| \ \geq\ \|\texttt{selects}(xs,\ ys1)\| \\ \|\texttt{selects([}x\ \texttt{|}\ xs\texttt{]},\ ys)\| \ \geq\ |ys| - |\texttt{[}x\ \texttt{|}\ xs\texttt{]}|. \end{cases}$$

As the second constraint suggests, we put

$$\|\texttt{selects}(xs,\ ys)\| = |ys| - |xs|.$$

It is worth noting that $\|\ \|$ range on integers. For $|ys| = |ys1| + 1$, we have:

$$\|\texttt{selects([}x\ \texttt{|}\ xs\texttt{]},\ ys)\| = |ys| - |xs| - 1 = |ys1| - |xs| = \|\texttt{selects}(xs,\ ys1)\|.$$

Finally, we can conclude that $semi\!-\vdash_t \{Pre_{\texttt{SELS}}\}$ SELECTS $\{Pre_{\texttt{SELS}}\}$ holds by the (expected) level mapping

$$\begin{aligned} |\texttt{selects}(xs,\ ys)| &= |ys| - |xs| + |xs| = |ys| \\ |\texttt{select}(x,\ xs,\ ys)| &= |xs|. \end{aligned}$$

The resulting level mapping is expresive even if one of the level mappings we started with was counter-intuitive because computed by a local analysis of the sub-program. □

6 Conclusion

In this paper an re-formulation is given of the proof method proposed in [PR96, Rug94] for proving (weak) total correctness of logic programs. The new relation $semi\!-\vdash_t$ is well-suited for modular verification of programs, preventing some inexpressiveness phenomena occurring with relation $\vdash_t$.

References

[AB94] K. R. Apt and R. Bol. Logic programming and negation: a survey. *Journal of Logic Programming*, 19-20:9–71, 1994.

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

[AP94] K. R. Apt and D. Pedreschi. Modular termination proofs for logic and pure Prolog programs. In G. Levi, editor, *Advances in Logic Programming Theory*, pages 183–229. Oxford University Press, 1994.

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

[Bez89] M. Bezem. Characterizing termination of logic programs with level mappings. In E. L. Lusk and R. A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 69–80. The MIT Press, 1989.

[Jia94] Y. Jiang. Ambivalent logic as the semantic basis fo metalogic programming: I. In P. Van Hentenryck, editor, *Proceedings of the International Conference on Logic Programming*, pages 387–401. MIT Press, June 1994.

[KJ95] M. Kalsbeek and Y. Jiang. A vademecum of Ambivalent logic. In K.R. Apt and F. Turini, editors, *Meta-logics and Logic Programming*, pages 27–56. The MIT Press, 1995.

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

[PR96] D. Pedreschi and S. Ruggieri. Verification of Logic Programs. Technical report, Dipartimento di Informatica, Università di Pisa, 1996. To appear.

[Rug94] S. Ruggieri. Metodi formali per lo sviluppo di programmi logici. Master's thesis, Dipartimento di Informatica, Università di Pisa, 1994. Tesi di Laurea, in Italian.

A Modularity

THEOREM 5.6
Proof. Given a clause in $ground_L(P \cup Q)$, we distinguish two cases:

Consider $A \leftarrow B_1, \ldots, B_n \in ground_L(Q)$. For $i \in [1, n]$, if

$$Pre_P \cup Pre_Q \models A \tag{3}$$

$$\text{and} \quad Post_P \cup Post_Q \models B_1, \ldots, B_{i-1}. \tag{4}$$

then (3) and *(iii)* imply $Pre_Q \models A$. Moreover, 4 and *(iv)* imply $Post_Q \models B_1, \ldots, B_{i-1}$. As $\vdash \{Pre_Q\}\ Q\ \{Post_Q\}$ holds by *(ii)*, we have $Pre_Q \models B_i$ and then $Pre_P \cup Pre_Q \models B_i$. In a similar way, one shows that $Pre_P \cup Pre_Q \models A$ and $Post_P \cup Post_Q \models B_1, \ldots, B_n$ imply $Post_P \cup Post_Q \models A$.

Consider now $A \leftarrow B_1, \ldots, B_n \in ground_L(P)$. Suppose for $i \in [1, n]$ that (3) and (4) hold. Since in Pre_Q only atoms whose predicate symbol is in Π_Q appear (by the assumption), and in Q there is no predicate symbol defined in P (as P extends Q), we have that (3) implies $Pre_P \models A$, and (4) and *(iv)* imply $Post_P \models B_1, \ldots, B_{i-1}$. As $\vdash \{Pre_P\}\ P\ \{Post_P\}$ holds by *(i)*, we have $Pre_P \models B_i$ and then $Pre_P \cup Pre_Q \models B_i$. In a similar way, one shows that $Pre_P \cup Pre_Q \models A$ and $Post_P \cup Post_Q \models B_1, \ldots, B_n$ imply $Post_P \cup Post_Q \models A$.

Suppose now $Post_Q$ is a well-supported interpretation, i.e. there exist $(W_Q, <_Q)$ and $|\ |_Q : B_L \rightarrow W$ such that

$$\forall A \in Post_Q \cap Pre_Q\ \exists A \leftarrow B_1, \ldots, B_n \in ground_L(P)\ \ \forall i \in [1, n] : Post \models B_i \wedge |A|_Q >_P |B_i|_Q$$

Without lack of generality, we can assume W_Q and W_P to have a least element, say $\perp_Q$ and $\perp_P$ respectively. We show that $Post_P \cup Post_Q$ is well-supported by means of the well-founded poset $(W_P \times W_Q, <)$ where

$$(x, y) < (u, v) \quad \text{iff} \quad x <_P u \vee (u \not<_P x \wedge y <_Q v)$$

and the function $|\ | : B_L \rightarrow W_P \times W_Q$

$$|A| = \begin{cases} (|A|_P, \perp_Q) & \text{if } rel(A) \text{ defined in } P \\ (\perp_P, |A|_Q) & \text{if } rel(A) \text{ defined in } Q \\ (\perp_P, \perp_Q) & \text{otherwise} \end{cases}$$

First, we need the following fact.

FACT 1 *In the hypothesys (iii,iv) we have*

$$(Post_P \cup Post_Q) \cap (Pre_P \cup Pre_Q)) = (Post_P|_{\Pi_P \setminus \Pi_Q} \cap Pre_P) \cup (Post_Q \cap Pre_Q)$$

Proof.

$$
\begin{array}{cl}
 & (Post_P|_{\Pi_P \setminus \Pi_Q} \cap Pre_P) \cup (Post_Q \cap Pre_Q) \\
= & \quad \{ \ Post_Q \cap (Pre_P \setminus Pre_Q) = \emptyset \text{ by the Assumption and } \textit{(iii)} \ \} \\
 & (Post_P|_{\Pi_P \setminus \Pi_Q} \cap Pre_P) \cup (Post_Q \cap (Pre_P \cup Pre_Q)) \\
= & \quad \{ \text{ Assumption } \} \\
 & (Post_P|_{\Pi_P \setminus \Pi_Q} \cap (Pre_P \cup Pre_Q)) \cup (Post_Q \cap (Pre_P \cup Pre_Q)) \\
= & \quad \{ \text{ Distributivity and } \textit{(iv)} \ \} \\
 & (Post_P \cup Post_Q) \cap (Pre_P \cup Pre_Q))
\end{array}
$$

□

Let us show that $Post_P \cup Post_Q$ is well-supported. Consider $A \in Post_P \cup Post_Q$. According to the equality stated by Fact 1, we distinguish two cases.

$A \in Post_Q \cup Pre_Q)$ Consider the clause $A \leftarrow B_1, \ldots, B_n$ given by the hypothesis that $Post_Q$ is well-supported. We have $Post_Q \models B_i$ for $i \in [1,n]$ and then $Post_P \cup Post_Q \models B_i$. Moreover

$$|A| = (\perp_P, |A|_Q) > (\perp_P, |B_i|_Q) \geq (\perp_P, \perp_Q)$$

Therefore, the conclusion $|A| > |B_i$ follows in either cases when B_i is defined in Q or it is not defined in $P \cup Q$.

$A \in Post_P|_{\Pi_P \setminus \Pi_Q} \cap Pre_P)$ Consider the clause $A \leftarrow B_1, \ldots, B_n$ given by the hypothesis. We have $Post_P \models B_i$ for $i \in [1,n]$ and then $Post_P \cup Post_Q \models B_i$. Moreover, for $i \in [1,n]$

- if B_i is defined in P then, by the hypothesis $|A|_P > |B_i|_P$ and then $|A| = (|A|_P, \perp_Q) > (|B_i|_P, \perp_Q) = |B_i|$;
- if B_i is defined in Q then $|A|_P > |B_i|_P \geq \perp_P$ implies $|A| = (|A|_P, \perp_Q) > (\perp_P, |B_i|_Q) = |B_i|$;
- otherwise, $|A|_P > |B_i|_P \geq \perp_P$ implies $|A| > (\perp_P, \perp_Q) = |B_i|$.

□

THEOREM 5.7

Proof. We consider two cases and concentrate only on decreasing of the level mapping.

Consider $A \leftarrow B_1, \ldots, B_n \in ground_L(Q)$. For $i \in [1,n]$ if $Pre_P \cup Pre_Q \models A$ and $Post_P \cup Post_Q \models B_1, \ldots, B_{i-1}$ then reasoning as in the proof of Theorem 5.6 we have

$$Pre_Q \models A \quad \text{and} \quad Post_Q \models B_1, \ldots, B_{i-1} \tag{5}$$

P extends Q implies $|B_i| = |B_i|_Q$ or $\|B_i\| = 0$ if respectively $rel(B_i)$ is or is not defined in Q. Therefore

$$
\begin{array}{ccl}
|A| & = & |A|_P \\
 & > & \quad \{ \ \vdash_t \{Pre_P\} \ P \ \{Post_P\} \text{ and } (5) \ \} \\
 & & |B_i|_P \\
 & \geq & \quad \{ \text{ Definition of } |\ | \ \} \\
 & & |B_i|
\end{array}
$$

Consider now $A \leftarrow B_1, \ldots, B_n \in ground_L(P)$. For $i \in [1,n]$ if $Pre_P \cup Pre_Q \models A$ and $Post_P \cup Post_Q \models B_1, \ldots, B_{i-1}$ then reasoning as in the proof of Theorem 5.6 we have

$$Pre_P \models A \quad \text{and} \quad Post_P \models B_1, \ldots, B_{i-1} \tag{6}$$

We distinguish two further cases

- if $rel(B_i)$ is defined in Q then $\vdash \{Pre_Q\} \ Q \ \{Post_Q\}$ and (6) imply $|A| = |A|_Q > |B_i|_Q = |B_i|$;
- otherwise, $rel(B_i)$ is not defined in $P \cup Q$ and then $|A| = |A|_Q > |B_i|_Q \geq 0 = |B_i|$.

To show the second part of the theorem, we recall from Theorem 4.3 that if $\vdash_t \{Pre\} \ P \ \{Post\}$ holds then $Post$ is well-supported w.r.t. P and Pre iff $T_P(Post) \supseteq Post \cap Pre$.

$$
\begin{array}{cl}
 & T_{P \cup Q}(Post_P \ \cup \ Post_Q) \\
\supseteq & \quad \{ \text{ monotonicity and compositionality of } T_P \ \} \\
 & T_P(Post_P) \cup T_Q(Post_Q)
\end{array}
$$

$$
\begin{array}{cl}
\supseteq & \{T_Q(Post_Q) \supseteq Post_Q \cap Pre_Q \text{ and } (iii)\} \\
& T_P(Post_P) \cup (Post_Q \cap (Pre_P \cup Pre_Q)) \\
\supseteq & \{\ (vi)\ \} \\
& (Post_P|_{\Pi_P \setminus \Pi_Q} \cap Pre_P) \cup (Post_Q \cap (Pre_P \cup Pre_Q)) \\
= & \{\ \text{Assumption}\ \} \\
& (Post_P|_{\Pi_P \setminus \Pi_Q} \cap (Pre_P \cup Pre_Q)) \cup (Post_Q \cap (Pre_P \cup Pre_Q)) \\
= & \{\ \text{Distributivity and } (iv)\ \} \\
& (Post_P \cup Post_Q) \cap (Pre_P \cup Pre_Q))
\end{array}
$$

□

THEOREM 5.8

Proof. We consider two cases and concentrate only on decreasing of the level mapping. First, we point out that $|\ |$ is well-defined by *(v)c*.

Consider $A \leftarrow B_1, \ldots, B_n \in ground_L(Q)$. For $i \in [1, n]$ if $Pre_P \cup Pre_Q \models A$ and $Post_P \cup Post_Q \models B_1, \ldots, B_{i-1}$ then reasoning as in the proof of Theorem 5.6 we have

$$Pre_Q \models A \quad \text{and} \quad Post_Q \models B_1, \ldots, B_{i-1} \tag{7}$$

We consider two cases

- if $rel(A) \simeq rel(B_i)$ then

$$
\begin{array}{rcl}
|A| & = & |A|_Q \\
& > & \{\ (ii) \text{ and } (7)\ \} \\
& & |B_i|_Q \\
& = & \{\ rel(B_i) \text{ defined in } Q\ \} \\
& & |B_i|
\end{array}
$$

- if $rel(A) \sqsupset rel(B_i)$ then

$$
\begin{array}{rcl}
|A| & = & |A|_Q \\
& \geq & \{\ (ii) \text{ and } (7)\ \} \\
& & |B_i|_Q \\
& \geq & \{\ rel(B_i) \text{ is not defined in } P\ \} \\
& & |B_i|
\end{array}
$$

Consider now $A \leftarrow B_1, \ldots, B_n \in ground_L(P)$. For $i \in [1, n]$ if $Pre_P \cup Pre_Q \models A$ and $Post_P \cup Post_Q \models B_1, \ldots, B_{i-1}$ then reasoning as in the proof of Theorem 5.6 we have

$$Pre_P \models A \quad \text{and} \quad Post_P \models B_1, \ldots, B_{i-1} \tag{8}$$

We distinguish three further cases

- if $rel(B_i)$ is defined in P and $rel(A) \simeq$ (resp., $\sqsupset$) $rel(B_i)$ then

$$
\begin{array}{rcl}
|A| & = & |A|_P + \|A\| \\
& \geq & \{\ (v)a\ \} \\
& & |A|_P + \|B_i\| \\
& > \text{ (resp., } \geq \text{)} & \{\ (i) \text{ and } (8)\ \} \\
& & |B_i|_P + \|B_i\| \\
& \geq & \{\ rel(B_i) \text{ is defined in } P\ \} \\
& & |B_i|
\end{array}
$$

- if $rel(B_i)$ is defined in Q then $rel(A) \sqsupset rel(B_i)$ and then

$$\begin{array}{rcl} |A| & = & |A|_P + \|A\| \\ & \geq & \quad \{ \text{ (v)}b \ \} \\ & & |B_i|_Q \\ & = & \quad \{ \ rel(B_i) \text{ defined in } Q \ \} \\ & & |B_i| \end{array}$$

- and, finally, if $rel(B_i)$ is not defined in $P \cup Q$ we have

$$\begin{array}{rcl} |A| & = & |A|_P + \|A\| \\ & \geq & \quad \{ \text{ (v)}c \ \} \\ & & 0 \\ & = & \quad \{ \ rel(B_i) \text{ is not defined in } P \cup Q \ \} \\ & & |B_i| \end{array}$$

$\square$