Specialising Logic Programs wrt Pre/Post Specifications[1]

Annalisa Bossi* and Sabina Rossi**

*Dipartimento di Matematica e Informatica, via Torino 155, 30173 Venezia, Italy
e-mail: bossi@dsi.unive.it
**Dipartimento di Matematica, via Belzoni 7, 35131 Padova, Italy
e-mail: sabina@math.unipd.it

Abstract

In this paper we introduce the concept of *specialised derivations.* They are intended to describe program behaviour whenever some constraints on procedure calls (preconditions) are assumed. Operational and fixpoint semantics, in the s-semantics style, are derived. They characterize successful derivations where only atoms satisfying the given precondition are selected.

A novel notion of *specialised partial correcteness* (*s.p.c.*) of a program with respect to pre/post specifications is introduced and a proof method is defined. The latter consists in one application of the specialised immediate consequence operator to the "relevant" part of the postcondition. Programs, which are proved to be s.p.c. wrt a given pre/post specification, can be transformed into partially correct ones.

1 Introduction

Techniques for proving correctness of a program with respect to a given pre/post specification are usually based on inductive methods [7, 4, 6, 2, 3]. As in any inductive proof, the choice of the inductive hypothesis is critical to carry out the proof. A stronger hypothesis is often necessary, namely that the precondition holds for all atoms selected in the derivations. Clearly, there exist programs which have a correct behaviour with respect to a pre/post specification even if some selected atoms do not satisfy the precondition. As an example, consider the program PERM [3]

$\texttt{perm}\,([\,],[\,]).$
$\texttt{perm}\,(X_s,[X|Y_s]) \leftarrow \texttt{append}\,(X1_s,[X|X2_s],X_s),\ \texttt{append}\,(X1_s,X2_s,Z_s),\ \texttt{perm}\,(Z_s,Y_s)$
$\texttt{append}\,([\,],X_s,X_s).$
$\texttt{append}\,([X|X_s],Y_s,[X|Z_s]) \leftarrow \texttt{append}\,(X_s,Y_s,Z_s).$

and the following pre/post specification.

$$\begin{array}{rcl} Pre & = & \{\texttt{append}\,(u,v,z) \mid u,v \text{ are terms and } z \text{ is a list of at most } n \text{ elements }\}\cup \\ & & \{\texttt{perm}\,(u,v) \mid u \text{ is a list of at most } n \text{ elements and } v \text{ is a variable }\} \\ Post & = & \{\texttt{append}\,(u,v,z) \mid u,v,z \text{ are lists, } z \text{ has at most } n \text{ elements and } z = u * v\}\cup \\ & & \{\texttt{perm}\,(u,v) \mid u,v \text{ are lists of at most } n \text{ elements }\} \end{array}$$

where $*$ is the list concatenation operator. The program is correct w.r.t. *Pre* and *Post.* However, in solving queries like $\texttt{perm}\,([a,b],X_s)$, we find procedure calls of the form $\texttt{append}\,(t_1,t_2,Z_s)$

[1]Extended Abstracts of LOPSTR'98, Eighth International Workshop on Logic-based Program Synthesis and Transformation, 15-19 June 1998, Manchester, UK. Technical Report Series, Department of Computer Science, University of Manchester, ISSN 1361-6161. Report number UMCS-98-6-1. http://www.cs.man.ac.uk/cstechrep/titles98.html

which do not satisfy the precondition since Z_s is a variable.
In [12, 13, 14], Naish provides a three-value semantics that allows to reason about correctness of programs without the need for undue retrictions on programming style.

In this paper, we introduce a notion of specialised partial correctness. It can be easly proved by applying a T_P-like operator to the "relevant" part of the postcondition. This notion refers to a definition of *specialised derivations* which is introduced in order to describe the intended behaviour of a program which meets its pre/post specification at each procedure call.

We show that the s-semantics approach [8, 9, 5] can be specialised to preconditions. The specialised denotations can be computed both by a top-down and a bottom-up construction.

Moreover, we develop a simple program transformation which transforms a s.p.c. program P into a specialised program P_{Pre}. We show that the specialised semantics of P is equivalent to the s-semantics of P_{Pre} and that in P_{Pre} all selected atoms in any derivation satisfy its precondition.

2 Specialised Derivations

The reader is assumed to be familiar with the terminology of and the basic results in the semantics of logic programs [1, 3, 11].

We denote by $\mathcal{T}$ the set of terms built on a finite set of *data conctructors* $\mathcal{C}$ and a denumerable set of *variable symbols* $\mathcal{V}$. A *substitution* is a mapping $\theta : \mathcal{V} \to \mathcal{T}$ such that the set $D(\theta) = \{X | \ \theta(X) \neq X\}$ (*domain* of θ) is finite. ϵ denotes the empty substitution. The *composition* $\theta\sigma$ of the substitutions θ and σ is defined as the functional composition, i.e., $\theta\sigma(X) = \sigma(\theta(X))$. A *renaming* is a substitution ρ for which there exists the inverse ρ^{-1} such that $\rho\rho^{-1} = \rho^{-1}\rho = \epsilon$. The pre-ordering $\leq$ (more general than) on substitutions is such that $\theta \leq \sigma$ iff there exists θ' such that $\theta\theta' = \sigma$. We say that θ and σ are *not comparable* if neither $\theta \leq \sigma$ nor $\sigma \leq \theta$. The result of the application of θ to a term t is an *instance* of t denoted by $t\theta$. We define $t \leq t'$ (t is more general than t') iff there exists θ such that $t\theta = t'$. The relation $\leq$ is a preorder. Let $\approx$ be the associated equivalence relation (*variance*). A substitution θ is a *unifier* of terms t and t' if $t\theta = t'\theta$. We denote by $mgu(t_1, t_2)$ any idempotent most general unifier (*mgu* in short) of t_1 and t_2. All the above definitions can be extended to other syntactic objects in the obvious way.

Atoms, queries, clauses and programs are defined as follows. Let $\mathcal{P}$ be a finite set of *predicate symbols*. An *atom* is an object of the form $p(t_1, \ldots, t_n)$ where $p \in \mathcal{P}$ is an n-ary predicate symbol and $t_1, \ldots, t_n \in \mathcal{T}$. A *query* is a possibly empty finite sequence of atoms $A_1, \ldots, A_m$. The empty query is denoted by $\Box$. A *clause* is a formula of the form $H \leftarrow \mathbf{B}$ where H is an atom and $\mathbf{B}$ is a query. H is called the *head* of the clause and $\mathbf{B}$ its *body*. When $\mathbf{B}$ is empty, $H \leftarrow \mathbf{B}$ is written $H \leftarrow$ and is called a *unit clause*. A *program* is a finite set of clauses. In the following, atoms are denoted by $A, B, C, H, \ldots$, queries by $Q, \mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots$, clauses by $c, d, \ldots$, and programs by P.

2.1 Interpretations

By the *extended Herbrand base* $\mathcal{B}^{\mathcal{E}}$ we mean the quotient set of all atoms with respect to $\approx$. The ordering induced by $\leq$ on $\mathcal{B}^{\mathcal{E}}$ will still be denoted by $\leq$. For the sake of simplicity, we represent the equivalence class of an atom A by A itself. An *interpretation* I is any subset of $\mathcal{B}^{\mathcal{E}}$.
Let us recall from [10] some definitions of useful operators on interpretations. Let I be an interpretation. The *upward closure* of I is the set

$$\lceil I \rceil = \{A \in \mathcal{B} \mid \exists A' \in I, A' \leq A\}.$$

The *set of minimal elements* of I is the set

$$Min(I) = \{A \in I \mid \forall A' \in I \text{ if } A' \leq A \text{ then } A = A'\}.$$

Example 2.1 Let I be $\{\texttt{append}(u, v, z) \mid u, v$ are terms and z is a list of at most n elements$\}$.

$$\begin{array}{rcl} Min(I) & = & \{\texttt{append}\ (X_s, Y_s, [\,]), \\ & & \ \texttt{append}\ (X_s, Y_s, [Z_1]), \\ & & \ \texttt{append}\ (X_s, Y_s, [Z_1, Z_2]), \\ & & \ \ldots \\ & & \ \texttt{append}\ (X_s, Y_s, [Z_1, Z_2, \ldots, Z_n])\} \end{array}$$

where $X_s, Y_s, Z_1, Z_2, \ldots, Z_n$ are distinct variables.

The notion of truth extends the classical one to account for non-ground formulas in the interpretations. So, if A is an atom then $I \models A$ iff $A \in \lceil I \rceil$; whereas, if Q is a query of the form $A_1, \ldots, A_n$ then $I \models Q$ iff $A_i \in \lceil I \rceil$ for all $i \in \{1, \ldots, n\}$. Note that $I \models A$ iff there exists $A' \in I$ such that $A' \leq A$; moreover, if $I \models A$ then for all A' such that $A \leq A'$, $I \models A'$.
We define the *set of minimal instances of an atom A satisfying I* as follows:

$$Min_I(A) = Min(\{A' \in \lceil A \rceil \mid I \models A'\}).$$

Example 2.2 Consider the interpretation I of the Example 2.1. Then

$$\begin{array}{rcl} Min_I(\texttt{append}\ ([\,], Y_s, Z_s)) & = & \{\texttt{append}\ ([\,], Y_s, [\,]), \\ & & \ \texttt{append}\ ([\,], Y_s, [Z_1]), \\ & & \ \texttt{append}\ ([\,], Y_s, [Z_1, Z_2]), \\ & & \ \ldots \\ & & \ \texttt{append}\ ([\,], Y_s, [Z_1, Z_2, \ldots, Z_n])\} \end{array}$$

where $Z_1, Z_2, \ldots, Z_n$ are distinct variables. Observe that although I is infinite, both $Min(I)$ and $Min_I(\texttt{append}\ ([\,], Y_s, Z_s))$ are finite.

Specialised derivations are defined upon the following notion of specialised unifier.

Definition 2.1 (Specialised Unifiers) Let I be an interpretation. A substitution θ is a *I-unifier* of t_1 and t_2 if it is a unifier and $I \models t_1\theta$. A *most general I-unifier* of t_1 and t_2, denoted by

$$mgu_I(t_1, t_2),$$

is any idempotent I-unifier θ such that for any other I-unifier θ', either $\theta \leq \theta'$ or θ and θ' are not comparable.

Example 2.3 Consider again the interpretation I of the Example 2.1. Then

$$mgu_I(\texttt{append}\ (U, V, W), \texttt{append}\ ([\,], X_s, X_s))$$

denotes both substitutions

$$\begin{array}{lcl}\theta_0 & = & \{U/[\,], V/X_s, W/X_s, X_s/[\,]\} \\ \theta_1 & = & \{U/[\,], V/X_s, W/X_s, X_s/[Z_1]\} \\ \theta_2 & = & \{U/[\,], V/X_s, W/X_s, X_s/[Z_1, Z_2]\} \\ & & \ldots \\ \theta_n & = & \{U/[\,], V/X_s, W/X_s, X_s/[Z_1, Z_2, \ldots, Z_n]\}.\end{array}$$

Note that the substitutions $\theta_0, \theta_1, \ldots, \theta_n$ are pairwise not comparable.

For simplicity, we will write $\theta = mgu_I(t_1, t_2)$ even if $mgu_I(t_1, t_2)$ is not uniquely determined.

It is well known that set inclusion does not adequately reflect the property of non-ground atoms of being representatives of all their ground instances. So, we refer to the partial ordering $\sqsubseteq$ on interpretations defined by Falaschi *et al.* [9] as follows:

- $I_1 \leq I_2$ iff $\forall A_1 \in I_1, \exists A_2 \in I_2$ such that $A_2 \leq A_1$.
- $I_1 \sqsubseteq I_2$ iff $(I_1 \leq I_2)$ and $(I_2 \leq I_1$ implies $I_1 \subseteq I_2)$.

Intuitively, $I_1 \leq I_2$ means that every atom verified by I_1 is also verified by I_2 (I_2 contains more *positive information*). Note that $\leq$ has different meanings for atoms and interpretations. $I_1 \sqsubseteq I_2$ means either that I_2 strictly contains more positive information than I_1 or (if the amount of positive information is the same) that I_1 expresses it by fewer elements that I_2 (I_2 is more redundant). The relation $\leq$ is a preorder, whereas the relation $\sqsubseteq$ is an ordering. If $I_1 \subseteq I_2$, then $I_1 \sqsubseteq I_2$. The set of all the interpretations $\mathcal{I}$ with the ordering $\sqsubseteq$ is a complete lattice, noted by $\langle \mathcal{I}, \sqsubseteq \rangle$, where $\mathcal{B}^{\mathcal{E}}$ is the top element and $\emptyset$ is the bottom element.

Example 2.4 Let us still consider the interpretation I of the Example 2.1. Then $I \leq Min(I)$, but also $Min(I) \sqsubseteq I$. Moreover, $Min_I(\texttt{append}\ ([\,], Y_s, Z_s)) \sqsubseteq Min(I)$.

2.2 Specialised Derivations

We are now ready to introduce our specialised derivations. They differ from standard SLD-derivations for the fact that at each derivation step specialised most general unifiers are computed instead of standard mgus.

Assume given a program P and an interpretation I. Let $\mathbf{A}, B, \mathbf{C}$ be a non empty query, c be a clause, $H \leftarrow \mathbf{B}$ be a variant of c variable disjoint with $\mathbf{A}, B, \mathbf{C}$. Suppose that B and H unify and $\theta = mgu_I(B, H)$. The query $(\mathbf{A}, \mathbf{B}, \mathbf{C})\theta$ is called an *I-resolvent of* $\mathbf{A}, B, \mathbf{C}$ *and* c *w.r.t.* B, *with an mgu* θ. We write then

$$\mathbf{A}, B, \mathbf{C} \xrightarrow{\theta}_{P,c,I} (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta$$

and call it *I-derivation step.* $H \leftarrow \mathbf{B}$ is called its *I-input clause.* If the program P is clear from the context or the clause c is irrelevant we drop a reference to them. An I-derivation is obtained by iterating I-derivation steps. A maximal sequence

$$\delta := Q_0 \xrightarrow{\theta_1}_{P,c_1,I} Q_1 \xrightarrow{\theta_2}_{P,c_2,I} \cdots Q_n \xrightarrow{\theta_{n+1}}_{P,c_{n+1},I} Q_{n+1} \cdots$$

of I-derivation steps is called an *I-derivation of* $P \cup \{Q_0\}$ where for every step the I-input clauses are standardized apart. If the program is clear from the context, we speak of an *I-derivation* of Q_0 and if clauses $c_1, \ldots, c_{n+1}, \ldots$ are irrelevant we drop the reference to them.

Consider a finite I-derivation $\delta := Q_0 \overset{\theta_1}{\Longrightarrow}_{P,c_1,I} Q_1 \cdots \overset{\theta_n}{\Longrightarrow}_{P,c_n,I} Q_n$ of a query $Q := Q_0$, also denoted by $\delta := Q_0 \overset{\theta}{\longmapsto}_I Q_n$ with $\theta = \theta_1 \cdots \theta_n$. If $Q_n = \Box$ then δ is called *successful.* The restriction of θ to the variables of Q is called a *I-computed answer substitution* of Q. If Q_n is non-empty and there is no I-input clause $H \leftarrow \mathbf{B}$ such that H unifies with the I-selected atom B of Q_n with a substitution $\theta = mgu_I(B, H)$, then δ is called *failed.*
Whenever I is omitted, we implicitly assume that $I = \mathcal{B}^{\mathcal{E}}$. It is easy to see that if I is the extended Herbrand base $\mathcal{B}^{\mathcal{E}}$, then I-derivations are indeed SLD-derivations.

Example 2.5 Consider the interpretation I of Example 2.1 and the program `APPEND`:
append $([\,], X_s, X_s)$.
append $([X|X_s], Y_s, [X|Z_s]) \leftarrow$ append (X_s, Y_s, Z_s).

Let $\theta_0, \theta_1, \ldots, \theta_n$ be the substitutions defined in the Example 2.3. Then

$$
\begin{array}{l}
\texttt{append}\ (U, V, W) \overset{\theta_0}{\longmapsto}_I \Box \\
\texttt{append}\ (U, V, W) \overset{\theta_1}{\longmapsto}_I \Box \\
\ldots \\
\texttt{append}\ (U, V, W) \overset{\theta_n}{\longmapsto}_I \Box
\end{array}
$$

are both successful I-derivations, whereas any I-derivation of append $([\,], \texttt{foo}, \texttt{foo})$ fails.

The following specialised Lifting Lemma holds.

Lemma 2.1 (Specialised Lifting Lemma) Let I be an interpretation and $\delta := Q\theta \overset{\sigma}{\longmapsto}_I \Box$ be a successful I-derivation of a query $Q\theta$. Then, there exists a successful I-derivation $\delta' := Q \overset{\sigma'}{\longmapsto}_I \Box$ where $\sigma' \leq \theta\sigma$.

2.3 Specialised Semantics

Based on the s-semantics approach [10], we can model various observable properies of specialised derivations, such as specialised computed answer substitutions, in terms of denotations which are Herbrand interpretations (extended with variables). The denotations can be computed both by a top-down construction (a specialised computed answer substitution semantics) and by a bottom-up construction (the least fixpoint of a specialised immediate consequences operator).

Definition 2.2 (Specialised Computed Answer Substitution Semantics) Let P be a program and I be an interpretation. The *I-computed answer substitution semantics of P* is

$$
\begin{array}{rl}
\mathcal{O}_I(P) = \{A \in \mathcal{B}^{\mathcal{E}} \mid & \exists p \in \mathcal{P}, \exists X_1, \ldots, X_n \text{ distinct variables in } \mathcal{V},\ \exists \theta, \\
& p(X_1, \ldots, X_n) \overset{\theta}{\longmapsto}_{P,I} \Box, \\
& A = p(X_1, \ldots, X_n)\theta\}.
\end{array}
$$

Note that if I is the extended Herbrand base $\mathcal{B}^{\mathcal{E}}$, then $\mathcal{O}_{\mathcal{B}^{\mathcal{E}}}(P)$ is the original s-semantics defined by Falaschi *et al.* in [9].

Example 2.6 Consider the program `APPEND` and the interpretation I of Example 2.1. Then

$$
\begin{array}{rcl}
\mathcal{O}_I(\mathtt{APPEND}) & = & \{\mathtt{append}\ ([\,],[\,],[\,]),\\
& & \ \mathtt{append}\ ([\,],[X_1],[X_1]),\\
& & \ldots\\
& & \ \mathtt{append}\ ([\,],[X_1,X_2,\ldots,X_n],[X_1,X_2,\ldots,X_n]),\\
& & \ldots\\
& & \ \mathtt{append}\ ([X_1],[\,],[X_1]),\\
& & \ \mathtt{append}\ ([X_1],[X_2],[X_1,X_2]),\\
& & \ldots\\
& & \ \mathtt{append}\ ([X_1],[X_2,\ldots,X_n],[X_1,X_2,\ldots,X_n]),\\
& & \ldots\\
& & \ \mathtt{append}\ ([X_1,X_2],[\,],[X_1,X_2]),\\
& & \ldots\}
\end{array}
$$

The bottom-up construction is given in terms of a suitable immediate consequences operator $T_{P,I}$ whose least fixpoint can be shown to be equivalent to $\mathcal{O}_I(P)$.

Definition 2.3 ($T_{P,I}$ Transformation) Let P be a program and I and J be interpretations.

$$
\begin{array}{rl}
T_{P,I}(J) = \{A \in \mathcal{B}^{\mathcal{E}} \mid & \exists A' \leftarrow B_1,\ldots,B_n \in P,\\
& \exists B'_1,\ldots,B'_n \text{ variant of atoms in } J \text{ and renamed apart},\\
& \exists \theta = mgu_I((B_1,\ldots,B_n),(B'_1,\ldots,B'_n)),\\
& A \in Min_I(A'\theta)\}.
\end{array}
$$

Note that if I is the extended Herbrand base $\mathcal{B}^{\mathcal{E}}$, then $T_{P,\mathcal{B}^{\mathcal{E}}}$ coincides with the S-transformation T_S defined in [9]. For any interpretation I, the transformation $T_{P,I}$ is monotonic and continuous in the complete lattice $\langle \mathcal{I}, \subseteq\ \rangle$. It is easy to prove that the least fixpoint $T_{P,I} \uparrow \omega$ of $T_{P,I}$ in the complete lattice $\langle \mathcal{I}, \subseteq\ \rangle$ coincides with the least fixpoint of $T_{P,I}$ in the complete lattice $\langle \mathcal{I}, \sqsubseteq\ \rangle$.

Definition 2.4 (Specialised Fixpoint Semantics) Let P be a program and I be an interpretation. The *I-fixpoint semantics of P* is defined as

$$\mathcal{F}_I(P) = T_{P,I} \uparrow \omega.$$

Theorem 2.1 (Equivalence of the Operational and Fixpoint Semantics) Let P be a program and I be an interpretation. Then $\mathcal{O}_I(P) = \mathcal{F}_I(P)$.

3 Assertion-based Characterization of Program Behaviour

We characterize program behaviour in terms of pre/post specifications as follows. Let P be a program and I and J be interpretations. We say that P *is specialised partially correct* (*s.p.c.*, for short) *wrt the precondition I and the postcondition J*, denoted by

$$\{I\}P\{J\},$$

iff for any query Q, $Q \overset{\theta}{\longmapsto}_{P,I} \Box$ implies $J \models Q\theta$.

Given a postcondition J, it always exists an interpretation I such that $\{I\}P\{J\}$: in fact, $\{J\}P\{J\}$ holds for any interpretation J.

Example 3.1 Consider the program APPEND and the interpretations

$$\begin{array}{lcl} \mathit{Pre} & = & \{\mathsf{append}\ (u,v,z) \mid u,v \text{ are terms and } z \text{ is a list of at most } n \text{ elements }\} \\ \mathit{Post} & = & \{\mathsf{append}\ (u,v,z) \mid u,v,z \text{ are lists, } z \text{ has at most } n \text{ elements and } z = u * v\} \end{array}$$

where $*$ is the list concatenation operator. Then, $\{\mathit{Pre}\}$ APPEND $\{\mathit{Post}\}$ holds.

We define the *strongest postcondition of P with respect to a precondition I*, denoted by $sp(P,I)$, as the smallest interpretation J wrt to $\sqsubseteq$ such that $\{I\}P\{J\}$. It is easy to prove that $\mathit{Min}(\mathcal{O}_I(P)) = sp(P,I)$. Moreover, $\{I\}P\{J\}$ holds iff $sp(P,J) = \mathit{Min}(T_{P,I}(sp(P,J)))$. This result provides a simple and efficient method to prove partial correctness of a program wrt to given pre/post specifications. It consists in the application of the specialised immediate consequences operator to the "relevant" part of the specified postcondition.

4 Specialised Programs

A s.p.c. P wrt a given pre/post specification I and J, i.e., such that $\{I\}P\{J\}$, can be easily transformed into a *specialised program* P_I which is s.p.c. wrt the precondition $\mathcal{B}^{\mathcal{E}}$ and and the postcondition J, i.e., such that the following assertion holds:

$$\{\mathcal{B}^{\mathcal{E}}\}P_I\{J\}.$$

Observe that if a program P satisfies $\{\mathcal{B}^{\mathcal{E}}\}P\{J\}$, then for any query Q and computed answer substitution θ of Q (using standard SLD-derivations), $J \models Q\theta$. Specialised programs are defined as follows.

Definition 4.1 (Specialised Program) Let P be a program and I be an interpretation. The I-program corresponding to P, denoted by P_I, is defined by

$$P_I \quad = \quad \{(H \leftarrow \mathbf{B})\gamma \mid \quad H \leftarrow \mathbf{B} \in P \text{ and } H\gamma \in \mathit{Min}_I(H)\}.$$

It is easy to prove that for any program P and interpretation I, $\mathcal{O}(P_I) = \mathcal{O}_I(P)$ and $\mathcal{F}(P_I) = \mathcal{F}_I(P)$. Moreover, the following proposition holds.

Proposition 4.1 Let P be a program and I be an interpretation such that $\{I\}P\{J\}$. Then $\{\mathcal{B}^{\mathcal{E}}\}P\{J\}$. Moreover, any selected atom in any derivation of a query Q in P_I satisfies I.

Example 4.1 Consider the program APPEND and the interpretations *Pre* and *Post* given in the Example 3.1. We can transform the program APPEND obtaining the following specialised program $\mathtt{APPEND}_{\mathit{Pre}}$.

```
append ([ ], [ ], [ ]).
append ([ ], [X1], [X1]).
append ([ ], [X1, X2], [X1, X2]).
...
append ([ ], [X1, X2, ..., Xn], [X1, X2, ..., Xn]).
append ([X1|Xs], Ys, [X1]) ← append (Xs, Ys, [ ]).
append ([X1|Xs], Ys, [X1, X2]) ← append (Xs, Ys, [X2]).
...
append ([X1|Xs], Ys, [X1, X2, ..., Xn]) ← append (Xs, Ys, [X2, ..., Xn]).
```

The assertion $\{\mathcal{B}\}$ $\texttt{APPEND}_{Pre}$ $\{Post\}$ holds, meaning that for any query Q, each successful derivation of $\texttt{APPEND}_{Pre} \cup \{Q\}$ produces a computed instance $Q\theta$ which is true in *Post*. Moreover, any call pattern occurring during the derivation satisfies the precondition *Pre*.

Observe that also the program `PERM` can be specialised wrt the pre/post specification given in the Introduction so that all the call-patterns satisfy the pre-condition *Pre*.

References

[1] K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[2] K. R. Apt. Program verification and prolog. In E. Börger, editor, *Specification and Validation Methods for Programming Languages and Systems*, pages 55–95. Oxford University Press, Oxford, 1995.

[3] K. R. Apt. *From Logic Programming to Prolog.* Prentice Hall, 1997.

[4] A. Bossi and N. Cocco. Verifying Correctness of Logic Programs. In G. Levi and M. Martelli, editors, *Proc. Sixth Int'l Conf. on Logic Programming*, pages 96–110. Springer-Verlag, Berlin, 1989.

[5] A. Bossi, M. Martelli, M. Gabrielli, and G. Levi. The *s*-semantics approach: theory and applications. *Journal of Logic Programming*, 19-20:149–197, 1994.

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

[7] W. Drabent and J. Maluszynski. Inductive Assertion Method for Logic Programs. *Theoretical Computer Science*, 59(1):133–155, 1988.

[8] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A new Declarative Semantics for Logic Languages. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 993–1005. The MIT Press, Cambridge, Mass., 1988.

[9] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[10] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A model-theoretic reconstruction of the operational semantics of logic programs. *Theoretical Computer Science*, 103(1):86–113, 1993.

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

[12] Lee Naish. Types and the intended meaning of logic programs. In Frank Pfenning, editor, *Types in logic programming*, pages 189–216. MIT Press, Cambridge, Massachusetts, 1992.

[13] Lee Naish. Verification of logic programs and imperative programs. In Jean-Marie Jacquet, editor, *Constructing logic programs*, pages 143–164. Wiley, Chichester, England, 1993.

[14] Lee Naish. A three-valued semantics for horn clause programs. Technical Report 98/4, Department of Computer Science, University of Melbourne, Melbourne, Australia, March 1998.