Specialising Logic Programs with respect to Call/Post Specifications

Annalisa Bossi and Sabina Rossi

Dipartimento di Matematica e Informatica, via Torino 155, 30173 Venezia, Italy
{bossi,srossi}@dsi.unive.it

Abstract. In this paper we present a program specialisation method which, given a call/post specification, transforms a logic program into a *weakly call-correct* one satisfying the post-condition.
The specialisation is applied to *specialised partially correct* programs. This notion is based on the definition of *specialised derivation* which is intended to describe program behaviour whenever some properties on procedure calls are assumed. Top-down and bottom-up semantics of specialised derivations are recalled.

1 Introduction

Specialisation methods allow to restrict a program to a narrower context of application. The aim is that of obtaining a more efficient program which behaves as the original one on the considered subdomain.

In the field of logic programming, the narrowed context is usually described by means of a set of queries of interest. Specialisation methods are based on the idea of partially evaluating program clauses [5, 16, 20, 21] with respect to this set of queries. The goal is to detect clauses which are redundant in the restricted context, or to specialise them preventing costly failing derivations. If the original program is correct with respect to a pre/post specification [4, 9, 10, 2], and the considered queries satisfy the precondition, then the correctness of the specialised program is ensured. Nothing is guaranteed on the queries which are not in the set of interest. They may succeed with "wrong" answers, produce a finite failure or an infinite computation. We simply do not care about them.

The specialisation that we propose in this paper restricts the search space of a program to the set of derivations satisfying the property that, at each step, the selected atom "respects" a given call-condition. The main feature of our specialisation is that if the original program is "correct" (we say, specialised partially correct) with respect to a call/post specification then the specialised program is (partially) correct for all queries. The execution of a query which does not satisfy the specification ends with a finite failure.
As an application, our approach allows us to support types without the need of augmenting programs with any kind of type declaration.
To give an intuition, consider the program `SUM` defined by:

$\texttt{sum}\,(X, 0, X) \leftarrow .$
$\texttt{sum}\,(X, s(Y), s(Z)) \leftarrow \texttt{sum}\,(X, Y, Z).$

The intention is that of defining the sum between natural numbers which are represented by the constants 0, $s(0)$, $s(s(0))$ and so on. However, there are queries like $\texttt{sum}\,(a, s(0), s(a))$ which succeed even if some arguments are not numerals. In order to avoid such "wrong" solutions, one can make use of type checking predicates defined by additional Prolog procedures [17–19].
Our specialisation can be used to solve this problem by restricting the program to a suitable context of application that is expressed in the form of a call/post specification. Similarly to classical pre/post specifications, a call/post specification consists of two parts, a call-condition and a post-condition. While the notion of post-condition is the standard one, the notion of call-condition used in this paper is not standard. It represents an invariant property that is required to be satisfied by atomic queries after the unification with the input clause.
Going back to the example above, consider the call-condition characterizing all atoms of **sum** where the last argument is

- either the constant 0,
- or a term of the form $s(\_)$.

Consider also the post-condition denoting all atoms of **sum** whose arguments are natural numbers with the third one being the sum of the first two. Hence,

$Call = \{\texttt{sum}\,(u, v, z) \mid u, v \text{ are terms and } z \text{ is either } 0 \text{ or a term of the form } s(\_)\}$
$Post = \{\texttt{sum}\,(u, v, z) \mid u, v, z \text{ are natural numbers and } z = u + v\}$

By applying our specialisation transformation we obtain the specialised program:

$\texttt{sum}\,(0, 0, 0) \leftarrow .$
$\texttt{sum}\,(s(X), 0, s(X)) \leftarrow .$
$\texttt{sum}\,(X, s(Y), s(Z)) \leftarrow \texttt{sum}\,(X, Y, Z).$

We do not enter into the details of the specialisation here. We just observe that it is applied to the head of clauses. The specialised program called with the query $\texttt{sum}\,(a, s(0), s(a))$ fails. Moreover, it produces correct answers for any non failing input query.

Summarizing, in this paper we present a program specialisation which, given a call/post specification, transforms a logic program into a *weakly call-correct* one satisfying the post-condition. More precisely, the specialised program meets the following requirements: for any execution,

- each selected atom unified with the head of the input clause satisfies the call-condition;
- and each computed instance of a query satisfies the post-condition.

The specialisation is applied to so-called *specialised partially correct* programs. This notion is a generalization of the well-known concept of *partially correct* program [11,4,9,10,2,3]. It is based on the definition of *specialised derivation* which is a derivation where all the selected atoms are instantiated in order to satisfy a given call-condition. Thus, a specialised partially correct program satisfies the property that all its successful specialised derivations produce answers satisfying the post-condition. Specialised derivations has been introduced in [7] where we show that they can be computed both by a top-down and a bottom-up construction, in the style of the s-semantics [12,13,6]. We recall here such semantics. The equivalence of these two semantics can be proven by using the standard semantics of specialised programs as a link between them. The specialised semantics is useful to reason on the notion of specialised partial correctness. In particular it allows us to provide a sufficient condition to prove that a program is specialised partially correct with respect to a given call/post specification. It consists in one application of the specialised immediate consequence operator to the post-condition.

The paper is organized as follows. Section 2 recalls some basic notations and concepts. In Section 3 specialised derivations and their semantics are presented. In Section 4 our program specialisation is introduced. Section 5 discusses the equivalence between the top-down and the fixpoint semantics of specialised derivations. In Section 6 we provide a method for verifying specialised partial correctness of programs. Finally, in Section 7 we discuss a meaningful example.

2 Preliminaries

The reader is assumed to be familiar with the terminology of and the basic results in the semantics of logic programs [1,3,15].

Let $\mathcal{T}$ be the set of terms built on a finite set of *data constructors* $\mathcal{C}$ and a denumerable set of *variable symbols* $\mathcal{V}$. Variable-free terms are called *ground.*
A *substitution* is a mapping $\theta : \mathcal{V} \rightarrow \mathcal{T}$ such that the set $D(\theta) = \{X \mid \theta(X) \neq X\}$ (*domain* of θ) is finite. For any expression E, we denote by $\theta_{|E}$ the restriction of θ to the variables in $Var(E)$. ϵ denotes the empty substitution.
The *composition* $\theta\sigma$ of the substitutions θ and σ is defined as the functional composition, i.e., $\theta\sigma(X) = \sigma(\theta(X))$. 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 a substitution θ to a term t is an *instance* of t denoted by $t\theta$. We define $t \leq t'$ iff there exists θ such that $t\theta = t'$. We say that t and t' are *not comparable* if neither $t \leq t'$ nor $t' \leq t$. The relation $\leq$ on terms is a preorder. We denote by $\approx$ the associated equivalence relation (*variance*). A substitution θ is a *unifier* of 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. The definitions above are extended to other syntactic objects in the obvious way.

2.1 Programs and Derivations

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 $\Box$. We use the convention adopted by Apt in [3] and use bold characters to denote sequences of atoms. A *clause* is a formula $H \leftarrow \mathbf{B}$ where H is an atom, called *head*, and $\mathbf{B}$ is a query, called *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.

Computations are constructed as sequences of "basic" steps. Consider a non empty query $\mathbf{A}, B, \mathbf{C}$ and a clause c. Let $H \leftarrow \mathbf{B}$ be a variant of c variable disjoint with $\mathbf{A}, B, \mathbf{C}$. Let B and H unify with mgu θ. The query $(\mathbf{A}, \mathbf{B}, \mathbf{C})\theta$ is called an *SLD-resolvent of* $\mathbf{A}, B, \mathbf{C}$ *and* c *w.r.t.* B, *with an mgu* θ. The atoms B and $B\theta$ are called the *selected atom* and the *selected atom instance*, respectively, of $\mathbf{A}, B, \mathbf{C}$. We write then

$$\mathbf{A}, B, \mathbf{C} \overset{\theta}{\Longrightarrow}_{P,c} (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta$$

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

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

of SLD-derivation steps is called an *SLD-derivation of* $P \cup \{Q_0\}$ provided that for every step the standardization apart condition holds, i.e., the input clause employed is variable disjoint from the initial query Q_0 and from the substitutions and the input clauses used at earlier steps.

The length of an SLD-derivation δ, denoted by $len(\delta)$, is the number of SLD-derivation steps in δ. We denote by $Sel(\delta)$ the set of all the selected atom instances, one for each derivation step, of δ. If P is clear from the context, we speak of an SLD-derivation of Q_0. SLD-derivations can be finite or infinite. Consider a finite SLD-derivation $\delta := Q_0 \overset{\theta_1}{\Longrightarrow}_{P,c_1} Q_1 \cdots \overset{\theta_n}{\Longrightarrow}_{P,c_n} Q_n$ of a query $Q := Q_0$, also denoted by $\delta := Q_0 \overset{\theta}{\longmapsto}_{P,c_1,\ldots,c_n} Q_n$ (or simply $\delta := Q_0 \overset{\theta}{\longmapsto} Q_n$) 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, denoted by $\theta_{|Q}$, is called a *computed answer substitution* (*c.a.s.*, in short) of Q and $Q\theta$ is called a *computed instance* of Q. If Q_n is non-empty and there is no input clause whose head unifies with its selected atom, then the SLD-derivation δ is called *failed*.

2.2 Interpretations

By the *extended Herbrand base* $\mathcal{B}^{\mathcal{E}}$ we mean the quotient set of all the 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 will represent the equivalence class of an atom A by A itself. An *interpretation* I is any subset of $\mathcal{B}^{\mathcal{E}}$.
We recall from [14] some definitions of useful operators on interpretations.

Definition 1. (Operators on Interpretations) *Let I be an interpretation. The* upward closure *of I is the set*

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

The set of ground atoms *of I is the set*

$$\lfloor I \rfloor = \{A \in I \mid A \textit{ is ground}\}.$$

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

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

Example 1. Let I be the set $\{\texttt{append}(u, v, z) \mid u, v$ are terms and z is a list of at most n elements$\}$. Then

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

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

Let us introduce the notation $[I]$ as a shorthand for $\lfloor\lceil I \rceil\rfloor$. Note that $[I]$ is the set of all ground instances of atoms in I. Moreover (see [14]), $\lceil I \rceil = \lceil Min(I) \rceil$ and $Min(\lceil I \rceil) = Min(I)$.

The notion of truth extends the classical one to account for non-ground formulas in the interpretations.

Definition 2. ($\models$) *Let I be an interpretation and A be an atom . Then*

$$I \models A \textit{ iff } A \in \lceil I \rceil.$$

Moreover, if $Q := A_1, \ldots, A_n$ is a query then

$$I \models Q \textit{ iff } A_i \in \lceil I \rceil \textit{ for all } i \in \{1, \ldots, n\}.$$

The following properties hold [14]: $I \models A$ iff there exists $A' \in I$ such that $A' \leq A$; moreover, if $I \models A$ then for all A' such that $A \leq A'$, $I \models A'$.

Definition 3. (Minimal Instances of an Atom Satisfying I) *Let I be an interpretation and A be an atom. The* set of minimal instances of A satisfying I *is*

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

Example 2. Consider the interpretation I of the Example 1. Then

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

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

The notion of specialised unifier is introduced. It is the basic concept upon which specialised derivations are defined.

Definition 4. (Specialised Unifiers) *Let I be an interpretation and A_1 and A_2 be atoms. A* I-unifier *of A_1 and A_2 is a substitution θ such that $A_1\theta = A_2\theta$ and $I \models A_1\theta$. A* most general I-unifier *of A_1 and A_2, denoted by*

$$mgu_I(A_1, A_2),$$

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.

For the sake of simplicity, we will write $\theta = mgu_I(A_1, A_2)$ even if $mgu_I(A_1, A_2)$ is not uniquely determined and can even denote an infinite set of substitutions.

Example 3. Consider again the interpretation I of the Example 1. Then

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

denotes the following substitutions

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

Note that the substitutions $\theta_0, \theta_1, \ldots, \theta_n$ are pairwise not comparable. Moreover, they are idempotent but not relevant with respect to the variables occurring in the two unifying atoms. In fact, they contain new variables, $\{Z_1, \ldots, Z_n\}$, which are introduced in order to satisfy I. We call these variables *place holders.* Note that the definition of specialised *most* general unifier imposes that they are pairwise disjoint and distinct from the variables occurring in the unifying atoms.

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, in this paper, we refer to the partial ordering $\sqsubseteq$ on interpretations defined by Falaschi *et al.* in [14] as below.

Definition 5. *Let I_1 and I_2 be interpretation.*

- *$I_1 \leq I_2$ 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 that the amount of positive information is the same and I_1 expresses it by fewer elements than I_2. The relation $\leq$ is a preorder, whereas the relation $\sqsubseteq$ is an ordering. Moreover, if $I_1 \subseteq I_2$, then $I_1 \sqsubseteq I_2$.

Example 4. Consider the interpretation I of the Example 1. Then $I \leq Min(I)$, but also $Min(I) \sqsubseteq I$. Moreover, $Min_I(\texttt{append}\ ([\,], Y_s, Z_s)) \sqsubseteq Min(I)$.

The set of all the interpretations $\mathcal{I}$ under the relation $\sqsubseteq$ is a complete lattice.

Proposition 1. *[14] The set of interpretations $\mathcal{I}$ with the ordering $\sqsubseteq$ is a complete lattice, noted by $\langle \mathcal{I}, \sqsubseteq \rangle$. $\mathcal{B}^{\mathcal{E}}$ is the top element and $\emptyset$ is the bottom element.*

3 Specialised Derivations and their Semantics

In this section we recall from [7] the notion of specialised derivation and show some properties. Top-down and a fixpoint semantics are presented.

3.1 Specialised Derivations

Given an interpretation I, a specialised derivation is an SLD-derivation where all the selected atoms are instantiated in order to satisfy the call-condition I. Specialised derivations are defined as SLD-derivations except that at each derivation step specialised most general unifiers are computed instead of usual mgus. In the following we assume given a program P.

Definition 6. *Let I be an interpretation. 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 $\mathbf{A}, B, \mathbf{C}$. Let B and H unify and $\theta = mgu_I(B, H)$ where the place holders of θ are disjoint from $\mathbf{A}, \mathbf{B}, \mathbf{C}$. 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 specialised mgu θ. The atoms B and $B\theta$ are called the I-*selected atom *and I-*selected atom instance, *respectively, of $\mathbf{A}, B, \mathbf{C}$. We write then*

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

*and call it I-*derivation step. *$H \leftarrow \mathbf{B}$ is its I-*input clause. *A maximal sequence*

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

*of I-derivation steps is called an I-*derivation of *$P \cup \{Q_0\}$ where for every step the I-standardization apart condition holds, i.e., the I-input clause c_i employed and the place holders of unifier θ_i are variable disjoint from the initial query Q_0 and from the substitutions and the input clauses used at earlier steps.*
We denote by $Sel(\delta)$ the set of all the I-selected atom instances of δ.
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 $\delta := Q_0 \overset{\theta}{\longmapsto}_{P,c_1,\ldots,c_n,I} Q_n$ (or simply $\delta := Q_0 \overset{\theta}{\longmapsto}_I Q_n$) 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 *(I-*c.a.s.*, in short) of Q and $Q\theta$ is called a I-*computed instance *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 $\theta = mgu_I(B, H)$, then δ is called* failed*.*

Whenever I is omitted, we 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 (resp. I-derivation steps) are indeed SLD-derivations (resp. SLD-derivation steps).

Example 5. Consider the interpretation I of Example 1 and the program `APPEND`:

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

Let $\theta_0, \theta_1, \ldots, \theta_n$ be the substitutions defined in the Example 3. Then

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

are successful I-derivations of the query $\texttt{append}\,(U,V,W)$. Note that any I-derivation of the query $\texttt{append}\,([\,], \texttt{foo}, \texttt{foo})$ fails.

Observe that, for any I-derivation δ, $Sel(\delta) \leq I$, i.e., for all $A \in Sel(\delta)$, $I \models A$. Moreover, any SLD-derivation δ satisfying $Sel(\delta) \leq I$ is an I-derivation.

The following relation between successful I and SLD-derivations holds.

Lemma 1. *[7] Let I be an interpretation and $\delta := Q \xmapsto{\theta}_{c_1,\ldots,c_n,I} \Box$ be a successful I-derivation of a query Q. Then, there exists a successful SLD-derivation $\delta' := Q\theta \xmapsto{\gamma}_{c_1,\ldots,c_n} \Box$ of $Q\theta$ with $Q\theta\gamma = Q\theta$ and $Sel(\delta') \leq Sel(\delta)$.*

The next result is useful to reason on I-derivations.

Lemma 2. *Let I be an interpretation and $\delta := Q \xmapsto{\theta}_I \Box$ be a successful I-derivation of a query $Q := A_1, \ldots, A_n$. Then, for all $j \in \{1, \ldots, n\}$, there exists a successful I-derivation $\delta_j := A_j\theta \xmapsto{\gamma_j}_I \Box$ where $A_j\theta\gamma_j = A_j\theta$.*

Proof. Let $\delta := Q \xmapsto{\theta}_I \Box$. By Lemma 1, there exists a successful SLD-derivation $\delta' := Q\theta \xmapsto{\gamma} \Box$ with $Q\theta\gamma = Q\theta$ and $Sel(\delta') \leq Sel(\delta)$. By properties of SLD-derivations (see [1, 8, 15]), for all $j \in \{1, \ldots, n\}$, there exists a successful SLD-derivation $\delta_j := A_j\theta \xmapsto{\gamma_j} \Box$ such that $A_j\theta\gamma_j = A_j\theta$ and $Sel(\delta_j) \leq Sel(\delta')$. Then, for all $j \in \{1, \ldots, n\}$, $Sel(\delta_j) \leq Sel(\delta)$. Moreover, since δ is an I-derivation, $Sel(\delta) \leq I$. Hence, for all $j \in \{1, \ldots, n\}$, $Sel(\delta_j) \leq I$ and then, by Definition 6, δ_j is a successful I-derivation $A_j\theta \xmapsto{\gamma_j}_I \Box$.

We show that any I-computed instance is true in I.

Lemma 3. *Let I be an interpretation and $\delta := Q \xmapsto{\theta}_I \Box$ be a successful I-derivation of a query Q. Then $I \models Q\theta$.*

Proof. Let $Q := A_1, \ldots, A_n$. We prove that for all $j \in \{1, \ldots, n\}$, $I \models A_j\theta$.
By Lemma 2, for all $j \in \{1, \ldots, n\}$, there exists $\delta_j := A_j\theta \overset{\gamma_j}{\longmapsto}_I \Box$ where $A_j\theta\gamma_j = A_j\theta$. Let H_j be the head of the I-input clause used in the first I-derivation step of δ_j. Let also $\gamma_j^1 = mgu_I(A_j\theta, H_j)$. By Definition 4, $I \models A_j\theta\gamma_j^1$ and by Definition 6, $A_j\theta\gamma_j^1 \leq A_j\theta\gamma_j$. Hence, by Definition 2, $I \models A_j\theta\gamma_j$.

The Lifting Lemma for SLD-derivations [15, 1] can be generalized to specialised derivations as follows.

Lemma 4. (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 $\sigma' \leq \theta\sigma$.*

Proof. By induction on $len(\delta)$.
Basis. Let $len(\delta) = 1$. In this case Q consists of only one atom B and

$$\delta := B\theta \overset{\sigma}{\Longrightarrow}_I \Box$$

where $H \leftarrow$ is the I-input clause and $\sigma = mgu_I(B\theta, H)$. Because of standardization apart, we can assume that $\theta_{|H} = \epsilon$. Then, $\theta\sigma$ is a I-unifier of B and H. Hence, there exists $\sigma' = mgu_I(B, H)$ such that $\sigma' \leq \theta\sigma$ and

$$\delta' := B \overset{\sigma'}{\Longrightarrow}_I \Box$$

is a successful I-derivation.
Induction step. Let $len(\delta) > 1$. Then $Q := \mathbf{A}, B, \mathbf{C}$ and

$$\delta := (\mathbf{A}, B, \mathbf{C})\theta \overset{\sigma_1}{\Longrightarrow}_I (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta\sigma_1 \overset{\sigma_2}{\longmapsto}_I \Box$$

where B is the I-selected atom of Q, $c := H \leftarrow \mathbf{B}$ is the first I-input clause, $\sigma_1 = mgu_I(B\theta, H)$ and $\sigma = \sigma_1\sigma_2$. Because of standardization apart, we can assume that $\theta_{|c} = \epsilon$. Then, $\theta\sigma_1$ is a I-unifier of B and H. Hence, there exists $\sigma_1' = mgu_I(B, H)$ such that $\sigma_1' \leq \theta\sigma_1$ and

$$(\mathbf{A}, B, \mathbf{C}) \overset{\sigma_1'}{\Longrightarrow}_I (\mathbf{A}, \mathbf{B}, \mathbf{C})\sigma_1'$$

is an I-derivation step. Let γ be a substitution such that $\sigma_1'\gamma = \theta\sigma_1$. By the inductive hypothesis, there exists a successful I-derivation

$$(\mathbf{A}, \mathbf{B}, \mathbf{C})\sigma_1' \overset{\sigma_2'}{\longmapsto}_I \Box$$

where $\sigma_2' \leq \gamma\sigma_2$. Therefore,

$$\delta' := (\mathbf{A}, B, \mathbf{C}) \overset{\sigma_1'}{\Longrightarrow}_I (\mathbf{A}, \mathbf{B}, \mathbf{C})\sigma_1' \overset{\sigma_2'}{\longmapsto}_I \Box$$

is a successful I-derivation. Let $\sigma' = \sigma_1'\sigma_2'$. Then,

$$\begin{aligned}
\sigma' &= \sigma_1'\sigma_2' && \text{(by definition of } \sigma'\text{)} \\
&\leq \sigma_1'\gamma\sigma_2 && \text{(since } \sigma_2' \leq \gamma\sigma_2\text{)} \\
&= \theta\sigma_1\sigma_2 && \text{(since } \sigma_1'\gamma = \theta\sigma_1\text{)} \\
&= \theta\sigma && \text{(by definition of } \sigma\text{).}
\end{aligned}$$

3.2 Specialised Top-down Semantics

We present below a top-down construction which computes the specialised semantics of logic programs. It models the computed answer substitutions of the specialised derivations.

Definition 7. (Specialised Computed Answer Substitution Semantics) *Let P be a program and I be an interpretation. The I-*computed answer substitution semantics *of P is*

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

Observe 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 [13].

Example 6. Consider the interpretation I of Example 1. Then

$$\begin{aligned}\mathcal{O}_I(\texttt{APPEND}) = \{ & \texttt{append}\ ([\,],[\,],[\,]), \\ & \texttt{append}\ ([\,],[X_1],[X_1]), \\ & \texttt{append}\ ([\,],[X_1,X_2],[X_1,X_2]), \\ & \ldots \\ & \texttt{append}\ ([\,],[X_1,X_2,\ldots,X_n],[X_1,X_2,\ldots,X_n]), \\ & \texttt{append}\ ([X_1],[\,],[X_1]), \\ & \texttt{append}\ ([X_1],[X_2],[X_1,X_2]), \\ & \ldots \\ & \texttt{append}\ ([X_1],[X_2,\ldots,X_n],[X_1,X_2,\ldots,X_n]), \\ & \texttt{append}\ ([X_1,X_2],[\,],[X_1,X_2]), \\ & \ldots \\ \}. \end{aligned}$$

Let us consider now the *success set* and the *non-ground success set* semantics formally defined in [13]. The corresponding specialised versions are defined below.

Definition 8. (Specialised Success Set Semantics) *Let P be a program and I be an interpretation. The I-*success set semantics *of P is defined by*

$$\mathcal{O}_{I,1}(P) = \{A \in \mathcal{B}^{\mathcal{E}} \mid A \textit{ is ground and } A \xmapsto{\gamma}_I \Box \textit{ where } A\gamma = A\}.$$

Note that if I is the extended Herbrand base $\mathcal{B}^{\mathcal{E}}$, then $\mathcal{O}_{\mathcal{B}^{\mathcal{E}},1}(P)$ is the standard semantics, which is equivalent to the least Herbrand model [22].

Definition 9. (Specialised Non-ground Success Set Semantics) *Let P be a program and I be an interpretation. The I-*non-ground success set semantics *of P is defined by*

$$\mathcal{O}_{I,2}(P) = \{A \in \mathcal{B}^{\mathcal{E}} \mid A \xmapsto{\gamma}_I \Box \textit{ where } A\gamma = A\}.$$

If I is equal to $\mathcal{B}^{\mathcal{E}}$, then $\mathcal{O}_{\mathcal{B}^{\mathcal{E}},2}(P)$ is the *set of atomic logical consequences* [8] of P.

It is easy to prove that the following relations hold: $\mathcal{O}_{I,1}(P) = \lfloor \mathcal{O}_I(P) \rfloor$ and $\mathcal{O}_{I,2}(P) = \lceil \mathcal{O}_I(P) \rceil$.

3.3 Specialised Fixpoint Semantics

We define an immediate consequence operator $T_{P,I}$ on the set of interpretations $\mathcal{I}$. Its least fixpoint has been shown to be equivalent to the specialised computed answer substitutions semantics $\mathcal{O}_I(P)$.

Definition 10. ($T_{P,I}$ Transformation) *Let P be a program and I and J be two interpretations.*

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

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 [13].

Proposition 2. (Monotonicity and Continuity of $T_{P,I}$) *For any interpretation I, transformation $T_{P,I}$ is monotonic and continuous in the complete lattice $\langle \mathcal{I}, \subseteq \rangle$.*

Definition 11. (Powers of $T_{P,I}$) *As usual, we define powers of transformation $T_{P,I}$ as follows:*

$$\begin{array}{ll} T_{P,I} \uparrow 0 & = \emptyset, \\ T_{P,I} \uparrow n+1 & = T_{P,I}(T_{P,I} \uparrow n), \\ T_{P,I} \uparrow \omega & = \bigcup_{n \geq 0}(T_{P,I} \uparrow n). \end{array}$$

Proposition 3. *For any interpretation I, $T_{P,I} \uparrow \omega$ is the least fixpoint of $T_{P,I}$ in the complete lattice $\langle \mathcal{I}, \sqsubseteq \rangle$.*

Proof. By Proposition 2, $T_{P,I} \uparrow \omega$ is the least fixpoint of $T_{P,I}$ with respect to set inclusion. Moreover, for any fixpoint J of $T_{P,I}$, $T_{P,I} \uparrow \omega \subseteq J$. Hence, by Definition 5, $T_{P,I} \uparrow \omega \sqsubseteq J$.

The specialised fixpoint semantics is formally defined as follows.

Definition 12. (Specialised Fixpoint Semantics) *Let P be a program and I be an interpretation. The I-*fixpoint semantics* of P is defined as*

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

4 Specialising Programs wrt Call/Post Specification

In this section we define a simple program transformation which given a call/post specification transforms a program into a weakly call-correct one satisfying the post-condition. The notion of weak call-correctness is formally defined as follows.

Definition 13. (Weak Call-correctness) *Let P be a program and I be an interpretation. We say that P is* weakly call-correct *with respect to the call-condition I iff for any query Q and SLD-derivation δ of $P \cup \{Q\}$, $Sel(\delta) \leq I$.*

Our specialisation is applied to so-called *specialised partially correct* programs which are introduced below.

4.1 Specialised Partially Correct Programs

In this section we introduce the concept of specialised partially correct program with respect to a given call/post specification. It provides a weaker notion of *partial correctness* where specialised derivations only are observed. In Section 6, a simple method for verifying specialised partial correctness will be presented.

Definition 14. *Let P be a program and Call and Post be interpretations. We say that P is* specialised partially correct *(*s.p.c.*, in short) with respect to the call-condition Call and the post-condition Post, noted*

$$\{Call\}P\{Post\}_{spec},$$

if and only if for any query Q,

$$Q \xrightarrow{\theta}_{P,Call} \square \text{ implies } Post \models Q\theta.$$

Observe that, if $Call = \mathcal{B}^{\mathcal{E}}$ then P is correct with respect to the post-condition *Post* according to the standard correctness definition. In this case, P is s.p.c. with respect to any call-condition *Call* and the post-condition *Post*.

Example 7. Consider the program `APPEND` and the interpretations

$Call = \{\texttt{append}\,(u, v, z) \mid u, v$ are terms and z is a list of at most n elements$\}$
$Post = \{\texttt{append}\,(u, v, z) \mid u, v, z$ are lists, z has at most n elements and $z = u * v\}$

where $*$ is the list concatenation operator. The program `APPEND` is s.p.c. with respect to the specification *Call* and *Post*, i.e., the following assertion holds:

$$\{Call\}\ \texttt{APPEND}\ \{Post\}_{spec}.$$

We define the strongest post-condition of a program P with respect to a given call-condition as follows.

Definition 15. (Strongest Post-condition) *Let P be a program. The* strongest post-condition *of P with respect to a call-condition I, noted $sp(P, I)$, is the smallest interpretation J with respect to $\sqsubseteq$ such that $\{I\}P\{J\}_{spec}$.*

The next Proposition characterizes the strongest post-condition of a program P wrt a call-condition I in terms of the I-c.a.s. semantics of P.

Proposition 4. *Let P be a program and I be an interpretation.*
Then, $Min(\mathcal{O}_I(P)) = sp(P, I)$.

Proof. We prove that $\{I\}P\{Min(\mathcal{O}_I(P))\}_{spec}$ holds, i.e., if $\delta := Q \xrightarrow{\theta}_I \square$ is a successful I-derivation of a query Q then $Min(\mathcal{O}_I(P)) \models Q\theta$.
Let $Q := A_1, \ldots, A_n$. By Lemma 2, for all $j \in \{1, \ldots, n\}$ there exists a successful I-derivation $\delta_j := A_j\theta \xrightarrow{\gamma_j}_I \square$ where $A_j\theta\gamma_j = A_j\theta$. For all $j \in \{1, \ldots, n\}$, let $p_j \in \mathcal{P}$ and $X_1, \ldots, X_n$ be distinct variables in $\mathcal{V}$ such that $p_j(X_1, \ldots, X_n) \leq$

$A_j\theta$. By Lemma 4, there exists a successful I-derivation $p_j(X_1,\ldots,X_n) \xmapsto{\theta_j}_I \Box$ where $p_j(X_1,\ldots,X_n)\theta_j \le A_j\theta$. By Definition 7, $p_j(X_1,\ldots,X_n)\theta_j \in \mathcal{O}_I(P)$. This proves that for all j, $Min(\mathcal{O}_I(P)) \models A_j\theta$ and then $Min(\mathcal{O}_I(P)) \models Q\theta$.

Further, for any interpretation J such that $\{I\}P\{J\}_{spec}$, $Min(\mathcal{O}_I(P)) \sqsubseteq J$. We first prove that $Min(\mathcal{O}_I(P)) \le J$, i.e., for all $A \in Min(\mathcal{O}_I(P))$ there exists $A' \in J$ such that $A' \le A$. Let $A \in Min(\mathcal{O}_I(P))$. By Definition 7, there exist $p \in \mathcal{P}$, $X_1,\ldots,X_n$ distinct variables in $\mathcal{V}$ and a substitution θ such that $p(X_1,\ldots,X_n) \xmapsto{\theta}_I \Box$ is a successful I-derivation and $A = p(X_1,\ldots,X_n)\theta$. By the hypothesis $\{I\}P\{J\}_{spec}$, $J \models A$. So, there exists $A' \in J$ such that $A' \le A$. Suppose now that $J \le Min(\mathcal{O}_I(P))$. Then $Min(\mathcal{O}_I(P)) \subseteq J$. Indeed, from the fact that both $Min(\mathcal{O}_I(P)) \le J$ and $J \le Min(\mathcal{O}_I(P))$, for all $A \in Min(\mathcal{O}_I(P))$ there exists $A' \in J$ and $A'' \in Min(\mathcal{O}_I(P))$ such that $A'' \le A' \le A$. By Definition 1 of operator Min, $A'' = A$ and then $A \in J$.

4.2 Specialised Programs

Any s.p.c. program P with respect to a given call/post specification I and J, i.e., such that $\{I\}P\{J\}_{spec}$ holds, can be transformed into a specialised program P_I which is weakly call-correct with respect to the call-condition I and satisfies the property $\{\mathcal{B}^{\mathcal{E}}\}P_I\{J\}_{spec}$. This means that for any query Q and SLD-derivation δ of $P_I \cup \{Q\}$ with computed answer substitution θ, $Sel(\delta) \le I$ and $J \models Q\theta$.

Specialised programs are obtained from the following program transformation.

Definition 16. (Specialised Program) *Let P be a program and I be an interpretation. The I*-program corresponding to *P, denoted by P_I, is defined as:*

$$\begin{aligned} P_I = \{(H \leftarrow \mathbf{B})\gamma \mid\ & H \leftarrow \mathbf{B} \in P \textit{ and } H\gamma \in Min_I(H) \\ & \textit{where } \gamma \textit{ is idempotent and} \\ & Var(\gamma) \cap Var(H \leftarrow \mathbf{B}) \subseteq Dom(\gamma) = Var(H)\}. \end{aligned}$$

The condition $Var(\gamma) \cap Var(H \leftarrow \mathbf{B}) \subseteq Dom(\gamma) = Var(H)$ allows us to avoid undesired bindings on the variables in the bodies.
Note that P_I may be an infinite program but it will be finite whenever $Min(I)$ is finite.

Example 8. Consider the program `APPEND` and the interpretations *Call* and *Post* given in the Example 7. The specialised program $\texttt{APPEND}_{Call}$ is defined by:

append $([\,],[\,],[\,])$.
append $([\,],[X_1],[X_1])$.
append $([\,],[X_1,X_2],[X_1,X_2])$.
...
append $([\,],[X_1,X_2,\ldots,X_n],[X_1,X_2,\ldots,X_n])$.
append $([X_1|X_s],Y_s,[X_1]) \leftarrow$ append $(X_s,Y_s,[\,])$.
append $([X_1|X_s],Y_s,[X_1,X_2]) \leftarrow$ append $(X_s,Y_s,[X_2])$.
...
append $([X_1|X_s],Y_s,[X_1,X_2,\ldots,X_n]) \leftarrow$ append $(X_s,Y_s,[X_2,\ldots,X_n])$.

It is easy to see that the assertion $\{\mathcal{B}^{\mathcal{E}}\}$ $\texttt{APPEND}_{Call}$ $\{Post\}_{spec}$ holds, meaning that for any query Q and successful SLD-derivation δ of $\texttt{APPEND}_{Call} \cup \{Q\}$ with computed answer substitution θ, $Post \models Q\theta$. Moreover, for any selected atom instance $A \in Sel(\delta)$, $Call \models A$. Hence, $Sel(\delta) \leq Call$.

Proposition 5. *Let P be a program and I be an interpretation with $\{I\}P\{J\}_{spec}$. Then, P_I is weakly call-correct with respect to I.*

Proof. Let δ be an SLD-derivation of P_I. We prove that for all $A \in Sel(\delta)$, $I \models A$. Indeed, for all $A \in Sel(\delta)$ there exists and SLD-derivation step

$$\mathbf{A}, B, \mathbf{C} \overset{\theta}{\Longrightarrow}_{P_I, \mathcal{B}^{\mathcal{E}}} (\mathbf{A}, \mathbf{B}\gamma, \mathbf{C})\theta$$

of δ where B is the selected atom, $(H \leftarrow \mathbf{B})\gamma$ is the input clause, $\theta = mgu(B, H\gamma)$ and $A = B\theta$. By Definition 16, $H \leftarrow \mathbf{B}$ is a variant of a clause of P such that $H\gamma \in Min_I(H)$. Hence, $I \models H\gamma\theta = B\theta = A$.

Proposition 6. *Let P be a program and I be an interpretation. Then, $\{I\}P\{J\}_{spec}$ implies $\{\mathcal{B}^{\mathcal{E}}\}P_I\{J\}_{spec}$.*

Proof. We need the following result.

Claim. [7] Let P be a program, $Q := \mathbf{A}, B, \mathbf{C}$ and I be an interpretation. If

$$\mathbf{A}, B, \mathbf{C} \overset{\theta}{\Longrightarrow}_{P_I, \mathcal{B}^{\mathcal{E}}} (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta$$

is an SLD-derivation step then there exists a substitution γ such that

$$\mathbf{A}, B, \mathbf{C} \overset{\gamma\theta}{\Longrightarrow}_{P,I} (\mathbf{A}, \mathbf{B}', \mathbf{C})\gamma\theta$$

is an I-derivation step where $\mathbf{B} = \mathbf{B}'\gamma$, $(\mathbf{A}, \mathbf{B}, \mathbf{C})\theta = (\mathbf{A}, \mathbf{B}', \mathbf{C})\gamma\theta$ and $\gamma_{|Q} = \epsilon$.

Suppose that $\{I\}P\{J\}_{spec}$ holds. We prove that for any query Q and successful SLD-derivation $\delta := Q \overset{\theta}{\longmapsto}_{P_I, \mathcal{B}^{\mathcal{E}}} \Box$, $J \models Q\theta$. In order to obtain this result, we prove that for any such δ, there exists a successful I-derivation $\delta' := Q\theta \overset{\sigma}{\longmapsto}_{P,I} \Box$ where $Q\theta\sigma = Q\theta$. The fact that $J \models Q\theta$ follows by the hypothesis $\{I\}P\{J\}_{spec}$. This is done by induction on $len(\delta)$.
Basis. Let $len(\delta) = 1$. In this case, Q consists of only one atom B and

$$\delta := B \overset{\theta}{\Longrightarrow}_{P_I, \mathcal{B}^{\mathcal{E}}} \Box.$$

By Claim 4.2, there exists a substitution γ such that

$$B \overset{\gamma\theta}{\Longrightarrow}_{P,I} \Box$$

is an I-derivation step and $\gamma_{|B} = \epsilon$. By Lemma 1, it follows that

$$\delta' := B\theta = B\gamma\theta \overset{\sigma}{\Longrightarrow}_{P,I} \Box$$

is a successful I-derivation of $B\theta$ where $B\theta\sigma = B\theta$.
Induction step. Let $len(\delta) > 1$. In this case $Q := \mathbf{A}, B, \mathbf{C}$ and

$$\delta := \mathbf{A}, B, \mathbf{C} \overset{\theta_1}{\Longrightarrow}_{P_I, \mathcal{B}^{\mathcal{E}}} (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta_1 \overset{\theta_2}{\longmapsto}_{P_I, \mathcal{B}^{\mathcal{E}}} \Box$$

with $\theta = \theta_1\theta_2$. By Claim 4.2 there exists a substitution γ such that

$$\mathbf{A}, B, \mathbf{C} \overset{\gamma\theta_1}{\Longrightarrow}_{P,I} (\mathbf{A}, \mathbf{B}', \mathbf{C})\gamma\theta_1$$

is an I-derivation step where $\mathbf{B} = \mathbf{B}'\gamma$, $(\mathbf{A}, \mathbf{B}, \mathbf{C})\theta_1 = (\mathbf{A}, \mathbf{B}', \mathbf{C})\gamma\theta_1$ and $\gamma_{|Q} = \epsilon$. Let $H \leftarrow \mathbf{B}'$ be the input clause and $\gamma\theta_1 = mgu(B, H)$. Then, by properties of substitutions [1] there exists a substitution σ_1 such that $\sigma_{1|Q\theta} = \epsilon$, $\sigma_{1|H\leftarrow\mathbf{B}'} = \gamma\theta$ and $\sigma_1 = mgu_I(B\theta, H)$. So,

$$(\mathbf{A}, B, \mathbf{C})\gamma\theta \overset{\sigma_1}{\Longrightarrow}_{P,I} (\mathbf{A}, \mathbf{B}', \mathbf{C})\gamma\theta\sigma_1$$

is an I-derivation step. Since $\gamma_{|Q} = \epsilon$ and $(\mathbf{A}, \mathbf{B}, \mathbf{C})\theta_1 = (\mathbf{A}, \mathbf{B}', \mathbf{C})\gamma\theta_1$, also

$$(\mathbf{A}, B, \mathbf{C})\theta \overset{\sigma_1}{\Longrightarrow}_{P,I} (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta\sigma_1$$

is an I-derivation step. By the inductive hypothesis,

$$(\mathbf{A}, \mathbf{B}, \mathbf{C})\theta \overset{\sigma_2}{\longmapsto}_{P,I} \Box$$

is a successful I-derivation where $(\mathbf{A}, \mathbf{B}, \mathbf{C})\theta\sigma_2 = (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta$. Moreover,

$$\begin{aligned}
(\mathbf{A}, \mathbf{B}, \mathbf{C})\theta &= (\mathbf{A}, \mathbf{B}', \mathbf{C})\gamma\theta && (\text{since } (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta_1 = (\mathbf{A}, \mathbf{B}', \mathbf{C})\gamma\theta_1)\\
&= \mathbf{A}\theta, \mathbf{B}'\gamma\theta, \mathbf{C}\theta && (\text{since } \gamma_{|Q} = \epsilon)\\
&= \mathbf{A}\theta, \mathbf{B}'\gamma\theta\sigma_1, \mathbf{C}\theta && (\text{since } \gamma\theta_{|H\leftarrow\mathbf{B}'} = \sigma_1 \text{ and } \sigma_1 \text{ is idempotent})\\
&= \mathbf{A}\theta, \mathbf{B}\theta\sigma_1, \mathbf{C}\theta && (\text{since } \mathbf{B} = \mathbf{B}'\gamma)\\
&= (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta\sigma_1 && (\text{since } \sigma_{1|Q\theta} = \epsilon).
\end{aligned}$$

Then,

$$\delta' := (\mathbf{A}, B, \mathbf{C})\theta \overset{\sigma_1}{\Longrightarrow}_{P,I} (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta\sigma_1 \overset{\sigma_2}{\longmapsto}_{P,I} \Box$$

is a successful I-derivation. Let $\sigma = \sigma_1\sigma_2$. Then

$$\begin{aligned}
Q\theta\sigma &= (\mathbf{A}, B, \mathbf{C})\theta\sigma_1\sigma_2 && (\text{by definition of } Q \text{ and of } \sigma)\\
&= \mathbf{A}\theta\sigma_2, B\theta\sigma_2, \mathbf{C}\theta\sigma_2 && (\text{since } \sigma_{1|Q\theta} = \epsilon)\\
&= \mathbf{A}\theta, B\theta\sigma_2, \mathbf{C}\theta && (\text{by inductive hypothesis})\\
&= \mathbf{A}\theta, B\theta, \mathbf{C}\theta && (\text{because of standardization apart})\\
&= Q\theta && (\text{by definition of } Q).
\end{aligned}$$

5 Equivalence of the Top-down and Fixpoint Semantics

In this section we discuss the equivalence between the specialised top-down semantics $\mathcal{O}_I(P)$ and the fixpoint semantics $\mathcal{F}_I(P)$. The reader is referred to [7] for more details. The proof follows from the fact that for any program P and interpretation I, both $\mathcal{O}_I(P) = \mathcal{O}(P_I)$ and $\mathcal{F}_I(P) = \mathcal{F}(P_I)$ hold.

The equivalence between $\mathcal{O}_I(P)$ and $\mathcal{O}(P_I)$ follows from the following result.

Proposition 7. *[7] Let P be a program and I be an interpretation. Then, there exists a successful SLD-derivation $\delta := Q \overset{\theta}{\longmapsto}_{P_I, \mathcal{B}^{\mathcal{E}}} \Box$ of a query Q iff there exists a successful I-derivation $\delta' := Q \overset{\theta'}{\longmapsto}_{P,I} \Box$ where $Q\theta = Q\theta'$.*

Theorem 1. *For any program P and interpretation I, $\mathcal{O}(P_I) = \mathcal{O}_I(P)$.*

Proof. Recall that $\mathcal{O}(P_I) = \mathcal{O}_{\mathcal{B}^{\mathcal{E}}}(P_I)$. The result follows from Proposition 7.

The next Proposition relates powers of transformations $T_{P_I, \mathcal{B}^{\mathcal{E}}}$ and $T_{P,I}$. It allows us to prove the equivalence between $\mathcal{F}_I(P)$ and $\mathcal{F}(P_I)$.

Proposition 8. *Let P be a program, I be an interpretation and A be an atom. Then for all $n > 0$, $A \in T_{P_I, \mathcal{B}^{\mathcal{E}}} \uparrow n$ iff $A \in T_{P,I} \uparrow n$.*

Theorem 2. *For any program P and interpretation I, $\mathcal{F}(P_I) = \mathcal{F}_I(P)$.*

Proof. Recall that $\mathcal{F}(P_I) = \mathcal{F}_{\mathcal{B}^{\mathcal{E}}}(P_I)$. The result follows from Proposition 8.

We are now in position to prove the equivalence between the specialised top-down and fixpoint semantics.

Theorem 3. (Equivalence of Specialised Top-Down and Fixpoint Semantics) *For any program P and interpretation I, $\mathcal{O}_I(P) = \mathcal{F}_I(P)$.*

Proof. By [13], $\mathcal{O}(P_I) = \mathcal{F}(P_I)$. The result follows from Theorems 1 and 2.

6 Verifying Specialised Partial Correctness

In this section, we show that the specialised partial correctness of a program with respect to a given call/post specification can be verified just by one application of the specialised immediate consequence operator $T_{P,I}$ to the given post-condition.

Let us first prove the following sufficient condition.

Lemma 5. *Let P be a program and I and J be two interpretations such that $sp(P, I) \sqsubseteq J$. Then, $\{I\}P\{J\}_{spec}$ holds.*

Proof. We prove that for any query Q and successful I-derivation $\delta := Q \overset{\theta}{\longmapsto}_I \Box$, $J \models Q\theta$. Let $Q := A_1, \ldots, A_n$. We show that for all $j \in \{1, \ldots, n\}$, $J \models A_j\theta$. By Lemma 2, for all $j \in \{1, \ldots, n\}$, there exists a successful I-derivation $\delta_j := A_j\theta \overset{\gamma_j}{\longmapsto}_I \Box$ where $A_j\theta\gamma_j = A_j\theta$. Let $p_j \in \mathcal{P}$ and $X_1, \ldots, X_n$ be distinct variables in $\mathcal{V}$ such that $p_j(X_1, \ldots, X_n) \leq A_j\theta$. By Lemma 4, for all j, there exists a successful I-derivation $p_j(X_1, \ldots, X_n) \overset{\theta_j}{\longmapsto}_I \Box$ where $p_j(X_1, \ldots, X_n)\theta_j \leq A_j\theta$. By Definition 7, $p_j(X_1, \ldots, X_n)\theta_j \in \mathcal{O}_I(P)$. Hence, by Proposition 4, $sp(P, I) \models A_j\theta$. The fact that $J \models A_j\theta$ follows from the hypothesis that $sp(P, I) \sqsubseteq J$.

The next Proposition provides a method for verifying specialised partial correctness with respect to a given call/post specification.

Proposition 9. *Let P be a program and I and J be interpretations. Then, $T_{P,I}(J) \sqsubseteq J$ implies $\{I\}P\{J\}_{spec}$.*

Proof. We first establish the following Claims. The proofs are given in [7].

Claim. Let P be a program and I and J be two interpretations.
Then $Min(T_{P,I}(J)) = Min(T_{P,I}(Min(J)))$.

Claim. For any interpretation I, the transformation $Min \circ T_{P,I}$ is monotonic in the complete lattice $\langle \mathcal{I}, \sqsubseteq \rangle$.

By Lemma 5, it is sufficient to prove that $T_{P,I}(J) \sqsubseteq J$ implies $sp(P, I) \sqsubseteq J$.
We first prove that for all $n \geq 0$, $Min(T_{P,I} \uparrow n) \sqsubseteq J$.
By induction on n
Basis. Let $n = 0$. Straightforward, since by Definition 11, $Min(T_{P,I} \uparrow 0) = \emptyset$.
Induction step. Let $n > 0$. In this case,

$$\begin{aligned}
Min(T_{P,I} \uparrow n) &= Min(T_{P,I}(T_{P,I} \uparrow n-1)) && \text{(by Definition 11)}\\
&= Min(T_{P,I}(Min(T_{P,I} \uparrow n-1))) && \text{(by Claim 6)}\\
&\sqsubseteq Min(T_{P,I}(J)) && \text{(by induction hypothesis and}\\
& && \text{Claim 6)}\\
&\sqsubseteq T_{P,I}(J) && \text{(by Definiton 1)}\\
&\sqsubseteq J && \text{(by hypothesis).}
\end{aligned}$$

It follows that $Min(\bigcup_{n\geq 0}(T_{P,I} \uparrow n)) \sqsubseteq J$.
In fact, $Min(\bigcup_{n\geq 0}(T_{P,I} \uparrow n)) \leq J$, i.e., for all $A \in Min(\bigcup_{n\geq 0}(T_{P,I} \uparrow n))$ there exists $A' \in J$ such that $A' \leq A$. This property follows from the fact that for all $A \in Min(\bigcup_{n\geq 0}(T_{P,I} \uparrow n))$ there exists $n > 0$ such that $A \in Min(T_{P,I} \uparrow n)$. As proved above, $Min(T_{P,I} \uparrow n) \sqsubseteq J$. Hence, by Definition 5, $Min(T_{P,I} \uparrow n) \leq J$, i.e., there exists $A' \in J$ such that $A' \leq A$.
Moreover, if $J \leq Min(\bigcup_{n\geq 0}(T_{P,I} \uparrow n))$ then $Min(\bigcup_{n\geq 0}(T_{P,I} \uparrow n)) \subseteq J$. In fact, since both $Min(\bigcup_{n\geq 0}(T_{P,I} \uparrow n)) \leq J$ and $J \leq Min(\bigcup_{n\geq 0}(T_{P,I} \uparrow n))$, for all $A \in Min(\bigcup_{n\geq 0}(T_{P,I} \uparrow n))$ there exists $A' \in J$ and $A'' \in Min(\bigcup_{n\geq 0}(T_{P,I} \uparrow n))$ such that $A'' \leq A' \leq A$. By Definition 1 of operator Min, $A'' = A$ and then $A \in J$. Therefore,

$$\begin{aligned}
sp(P, I) &= Min(\mathcal{O}_I(P)) && \text{(by Proposition 4)}\\
&= Min(\mathcal{F}_I(P)) && \text{(by Theorem 3)}\\
&= Min(T_{P,I} \uparrow \omega) && \text{(by Definition 12)}\\
&= Min(\textstyle\bigcup_{n\geq 0}(T_{P,I} \uparrow n)) && \text{(by Definition 11)}\\
&\sqsubseteq J && \text{(as proved above).}
\end{aligned}$$

7 An Example

In this section we illustrate by means of an example the specialisation method defined in Section 4.2. Consider the program **FRONT** [3] defined below,

$$
\begin{aligned}
&\texttt{front}\ (\texttt{void}, [\,]).\\
&\texttt{front}\ (\texttt{tree}\ (X, \texttt{void}, \texttt{void}), [X]).\\
&\texttt{front}\ (\texttt{tree}\ (X, L, R), X_s) \leftarrow \texttt{nel\_tree}\ (\texttt{tree}\ (X, L, R)),\\
&\qquad\qquad\qquad\qquad\qquad\quad \texttt{front}\ (L, L_s),\\
&\qquad\qquad\qquad\qquad\qquad\quad \texttt{front}\ (R, R_s),\\
&\qquad\qquad\qquad\qquad\qquad\quad \texttt{append}\ (L_s, R_s, X_s).\\
&\texttt{nel\_tree}\ (\texttt{tree}\ (\_, \texttt{tree}\ (\_, \_, \_), \_)).\\
&\texttt{nel\_tree}\ (\texttt{tree}\ (\_, \_, \texttt{tree}\ (\_, \_, \_))).
\end{aligned}
$$

augmented by the program **APPEND**. It computes the frontier of a binary tree, i.e., it is correct with respect to the post-condition

$$
\begin{aligned}
Post = \{&\texttt{front}\ (t, l) \mid l \text{ is the frontier of the binary tree } t\} \cup \{\texttt{nel\_list}\ (t) \mid t \text{ is a}\\
&\text{term}\} \cup \{\texttt{append}\ (u, v, z) \mid u, v, z \text{ are lists and } z = u * v\}
\end{aligned}
$$

where $*$ is the list concatenation operator.
The auxiliary relation `nel_tree` is used to enforce that a tree is a non-empty, non-leaf tree, i.e., that it is a term of the form `tree (x, left, right)` where either `left` or `right` does not equal `void`.
Observe that the simpler program that is obtained by removing the first atom $\texttt{nel\_tree}\ (\texttt{tree}(X, L, R))$ in the body of the third clause and by discarding the relation `nel_tree` is indeed incorrect. In fact, as shown in [3], the query $\texttt{front}(\texttt{tree}(X, \texttt{void}, \texttt{void}), X_s)$ would yield two different answers: $\{X_s/[X]\}$ by means of the second clause and $\{X_s/[\,]\}$ by means of the third clause.
Suppose that our application domain consists of the set of binary trees whose left subtrees are all leaves. This information can be expressed by means of the following call-condition:

$$
\begin{aligned}
Call = \{&\texttt{front}\ (t, l) \mid t \text{ is either the empty tree or a leaf or a term of the form}\\
&\texttt{tree}(u, r, s) \text{ where } r \text{ is a leaf and } u, s \text{ and } l \text{ are terms}\} \cup\\
&\{\texttt{nel\_list}\ (t) \mid t \text{ is a term}\} \cup \{\texttt{append}\ (u, v, z) \mid u, v, z \text{ are terms}\}
\end{aligned}
$$

Note that, since the program is correct with respect to *Post*, then it is also s.p.c. with respect to the call/post specification *Call* and *Post*, i.e.,

$$\{Call\}\texttt{FRONT}\{Post\}_{spec}.$$

That can be also proven by computing $T_{\texttt{FRONT}, Call}(Post)$. We obtain

$$
\begin{aligned}
T_{\texttt{FRONT}, Call}(Post) = &\{\texttt{front}\ (\texttt{void}, [\,]),\ \texttt{front}\ (\texttt{tree}\ (X, \texttt{void}, \texttt{void}), [X])\} \cup\\
&\{\texttt{front}\ (t, l) \mid t \text{ is a binary tree whose left subtree is a}\\
&\text{leaf and } l \text{ is the frontier of } t\} \cup\\
&\{\texttt{nel\_list}\ (t) \mid t \text{ is a non-empty, non-leaf tree }\} \cup\\
&\{\texttt{append}\ (u, v, z) \mid u, v, z \text{ are lists and } z = u * v\}.
\end{aligned}
$$

Then the program can be specialised into a weakly call-correct program $\mathtt{FRONT}_{Call}$. Consider the Definition 16. Observe that for all head H, different from the third clause head, $Min_{Call}(H) = H$, whereas, $Min_{Call}(\mathtt{front}(\mathtt{tree}\ (X, L, R), X_s)) = \{\mathtt{front}(\mathtt{tree}(X, \mathtt{void}, \mathtt{void}), X_s),\ \mathtt{front}(\mathtt{tree}(X, \mathtt{tree}(L, \mathtt{void}, \mathtt{void}), R), X_s)\}$. The specialisation results in the program $\mathtt{FRONT}_{Call}$ defined by:

$\mathtt{front}\ (\mathtt{void}, [\,])$.
$\mathtt{front}\ (\mathtt{tree}\ (X, \mathtt{void}, \mathtt{void}), [X])$.
$\mathtt{front}\ (\mathtt{tree}\ (X, \mathtt{void}, \mathtt{void}), X_s) \leftarrow \mathtt{nel\_tree}\ (\mathtt{tree}\ (X, \mathtt{void}, \mathtt{void})),$
$\qquad \mathtt{front}\ (\mathtt{void}, L_s),$
$\qquad \mathtt{front}\ (\mathtt{void}, R_s),$
$\qquad \mathtt{append}\ (L_s, R_s, X_s)$.
$\mathtt{front}\ (\mathtt{tree}\ (X, \mathtt{tree}\ (L, \mathtt{void}, \mathtt{void}), R), X_s) \leftarrow$
$\qquad \mathtt{nel\_tree}\ (\mathtt{tree}\ (X, \mathtt{tree}\ (L, \mathtt{void}, \mathtt{void}), R)),$
$\qquad \mathtt{front}\ (\mathtt{tree}\ (L, \mathtt{void}, \mathtt{void}), L_s),$
$\qquad \mathtt{front}\ (R, R_s),$
$\qquad \mathtt{append}\ (L_s, R_s, X_s)$.

augmented by definitions of the relations nel_tree and append.
Now by unfolding we obtain

$\mathtt{front}\ (\mathtt{void}, [\,])$.
$\mathtt{front}\ (\mathtt{tree}\ (X, \mathtt{void}, \mathtt{void}), [X])$.
$\mathtt{front}\ (\mathtt{tree}\ (\_,\ \mathtt{tree}\ (L, \mathtt{void}, \mathtt{void}), R), [L|R_s]) \leftarrow\ \mathtt{front}\ (R, R_s)$.

where both the relation nel_tree and the relation append are not used.

Acknowledgements This work was partly supported by the Italian MURST with the National Research Project 9701248444-004 on "Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software".

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, N. Cocco, and S. Dulli. A Method for Specializing Logic Programs. *ACM Transactions on Programming Languages and Systems*, 12(2):253–302, 1990.
6. 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.

7. A. Bossi and S. Rossi. Specialised semantics of logic programs. Technical Report CS-98-11, Dipartimento di Matematica e Informatica, Università di Venezia, 1998.
8. K. L. Clark. Predicate logic as a computational formalism. Research Report DOC 79/59, Imperial College, Department of Computing, London, 1979.
9. L. Colussi and E. Marchiori. Proving correctness of logic programs using axiomatic semantics. In *Proc. Eighth Int'l Conf. on Logic Programming*, pages 629–644. The MIT Press, Cambridge, Mass., 1991.
10. P. Deransart. Proof methods of declarative properties of definite programs. *Theoretical Computer Science*, 118:99–166, 1993.
11. W. Drabent and J. Maluszynski. Inductive Assertion Method for Logic Programs. *Theoretical Computer Science*, 59(1):133–155, 1988.
12. 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.
13. 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.
14. 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.
15. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second edition.
16. J. W. Lloyd and J. C. Shepherdson. Partial Evaluation in Logic Programming. *Journal of Logic Programming*, 11:217–242, 1991.
17. 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.
18. Lee Naish. Verification of logic programs and imperative programs. In Jean-Marie Jacquet, editor, *Constructing logic programs*, pages 143–164. Wiley, Chichester, England, 1993.
19. 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.
20. A. Pettorossi and M. Proietti. A theory of logic program specialization and generalization for dealing with input data properties. In *Proceedings of Partial Evaluation, International Seminar*, volume 1110 of *Lecture Notes in Computer Science.* Springer-Verlag, 1996.
21. A. Takeuchi and K. Furukawa. Partial evaluation of Prolog programs and its application to metaprogramming. In *Information Processing 86*, pages 415–420, 1986.
22. M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.