A Predicate Transformer for Unification

Livio Colussi
Dipartimento di Matematica Pura ed Applicata
Università di Padova
Via Belzoni 7, 35131 Padova, Italy
colussi@pdmat1.unipd.it

Elena Marchiori
Centre for Mathematics and Computer Science
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
elena@cwi.nl
and
Dipartimento di Matematica Pura ed Applicata
Università di Padova
Via Belzoni 7, 35131 Padova, Italy

Abstract

In this paper we study unification as predicate transformer. Given a unification problem expressed as a set of sets of terms $\mathcal{U}$ and a predicate P, we are interested in the strongest predicate R (w.r.t. the implication) s.t. if P holds before the unification of $\mathcal{U}$ then R holds when the unification is performed. We introduce a Dijkstra-style calculus that given P and $\mathcal{U}$ computes R. We prove the soundness, completeness and termination of the calculus. The predicate language considered contains monotonic predicates together with some non-monotonic predicates like var, $\neg ground$, $share$ and $\neg share$. This allows to use the calculus for the static analysis of run-time properties of Prolog programs.

1 Introduction

The standard view of logic programming is declarative, i.e. a program describes some predicate or function without referring to the way it will be computed. Nevertheless computational aspects become fundamental for the study of run-time properties of Prolog programs, like the actual form of the arguments of a goal before and after its call. In Prolog unification is the main computational mechanism since it produces the value of the variables during the execution of a goal in a program. To study its effect on the values of variables we study unification by means of predicate transformers. The use of predicate transformers for semantic analysis has been studied in the setting of imperative programming: it

was advocated by Floyd [5] and by Dijkstra [3] for program verification. The use of predicate transformers in the framework of logic programming is new. Given a unification problem expressed by a set of sets of terms $\mathcal{U}$, we introduce the predicate transformer $sp.\mathcal{U}$ such that $sp.\mathcal{U}.P$ is semantically equivalent to the strongest predicate R (w.r.t. implication) s.t. if P holds before the unification of $\mathcal{U}$, then R holds when the unification is performed. We show that $sp.\mathcal{U}.P$ could be computed in one step if P were a monotonic predicate. Since our aim is to infer run-time properties of Prolog programs, then the predicate language considered contains also non-monotonic predicates like *var* or *share*. For this reason a careful analysis of some intermediate steps of the unification process is necessary. This yelds to a non-trivial system of syntactic rules to compute $sp.\mathcal{U}.P$. The soundness, completeness and termination of the system is proved. The calculus can be used to infer run-time properties of logic programs. In Cousot and Cousot's original paper on abstract interpretation of imperative programs [2] everything was couched in terms of predicate transformers. Predicate transformers were used to define deductive semantics. Deductive semantics was used to design approximate program analysis frameworks. To propose a similar approach for logic programs we need the correspondent of program point for a logic program. In [7] Nilsson introduced a scheme for inferring run-time properties of logic programs based on a semantic description of logic programs that uses the concept of program point. We will show that the predicate transformer *sp* can be easily cast in such a theory.

The rest of the paper is organized as follows. The next section contains some preliminaries and introduces the predicate transformer $sp.\mathcal{U}$. Section 3 introduces the transformation rules to compute $sp.\mathcal{U}.P$. In section 4 the soundness, completeness and termination of the calculus are proved. In section 5 we illustrate the use of the calculus for defining a forward semantics of Prolog programs.

2 Unification as Predicate Transformer

The computational meaning of unification in Prolog relies on the concept of substitution. A *substitution* is a mapping from variables to terms such that $dom(\vartheta) \stackrel{\text{def}}{=} \{v \mid v\vartheta \neq v\}$ is finite. The notion of unification can be given w.r.t. a set of sets of terms [4] or w.r.t. a set of equations [6]. We choose the first approach. Let $\mathcal{U}$ be a finite set of sets of terms. A *unifier* for $\mathcal{U}$ is a substitution ϑ such that every set in $\mathcal{U}$, under the application of ϑ, becomes a singleton, i.e. $\forall S \in \mathcal{U}\ \forall t, t' \in S\ (t\vartheta = t'\vartheta)$. A *most general unifier* for $\mathcal{U}$ is a unifier ϑ such that for every unifier σ there exists a substitution γ such that $\vartheta\gamma = \sigma$. The *set of idempotent most general unifiers* for $\mathcal{U}$ will be denoted by $mgu(\mathcal{U})$. The operational meaning of $\mathcal{U}$ can be described as the partial function $\lambda\alpha.\alpha\mu$, where α is a substitution and μ is a fixed mgu in $mgu(\mathcal{U}\alpha)$; clearly $\lambda\alpha.\alpha\mu$ is undefined if $mgu(\mathcal{U}\alpha) = \emptyset$. We study unification by means of the predicate transformer $sp.\mathcal{U}$ (where *sp* stands for strongest postcondition [5]) with the following operational

meaning.

Definition 2.1 *sp.$\mathcal{U}$.P* is true in precisely those substitutions $\alpha\mu$ such that $P\alpha$ is true and $\mu \in mgu(\mathcal{U}\alpha)$.

The choice to represent the unification process as set of sets of terms is motivated by the following observations:
$mgu(\{\{f(t_1,\ldots,t_n), f(s_1,\ldots,s_n)\}\}) = mgu(\{\{t_1,s_1\},\ldots,\{t_n,s_n\}\})$ and
$mgu(\{S_1,\ldots,S_n\}) = mgu(\{S_1 \cup S_2, S_3,\ldots,S_n\})$ if $S_1 \cap S_2 \neq \emptyset$.
These two equalities will be used in our calculus for *sp.$\mathcal{U}$* and they clearly lead to consider sets of sets of terms. For sake of clarity, we use double square brackets to enclose sets of terms $S = [\![t_1,\ldots,t_m]\!]$ and braces to enclose sets of sets of terms $\mathcal{U} = \{S_1,\ldots,S_n\}$.

We call a predicate P *monotonic* if it is (semantically) invariant under instantiation, that is for all substitutions α,β if $P\alpha$ is true then $P\alpha\beta$ is true. Now let $\mathcal{U}$ be $\{[\![t^1_1,\ldots,t^1_{n_1}]\!],\ldots,[\![t^m_1,\ldots,t^m_{n_m}]\!]\}$: we denote by U the predicate $((t^1_1 = \ldots = t^1_{n_1}) \wedge \ldots \wedge (t^m_1 = \ldots = t^m_{n_m}))$. Then the following lemma holds.

Lemma 2.2 *Let P be a monotonic predicate. Then $P \wedge U$ is equivalent to sp.$\mathcal{U}$.P.*

Proof. Let α be s.t. $P\alpha$ is true and let $\mu \in mgu(\mathcal{U}\alpha)$. Then $U\alpha\mu$ is true and from P monotonic it follows that $P\alpha\mu$ is true.
Viceversa let α be s.t. $(P \wedge U)\alpha$ is true. Then $P\alpha$ is true and $\epsilon \in mgu(\mathcal{U}\alpha)$. So by Definition 2.1 $(sp.\mathcal{U}.P)\alpha$ is true. □

Lemma 2.2 allows to compute *sp.$\mathcal{U}$.P* when P is a monotonic predicate.

2.1 The Language

However we are interested also in properties that describe the structure of terms, like *var* or $\neg ground$, since we want to use the predicate transformer to infer run-time properties of logic programs. Thus we introduce the language $\mathcal{A}$ defined on the alphabet containing the following classes of symbols:

- a countable set VAR of *variables*;
- a set FUN of *functions*;
- a set $PRED = Pred \cup \{free, var, \neg ground, share, \neg share, inst\}$ of *predicate* symbols where $Pred$ is a finite set of monotonic predicate symbols s.t. $=$, *ground*, $\neg var$, $\prec$, $\preceq$, *invar* are in $Pred$;
- the *connectives* $\wedge$ and $\vee$;
- the *existential quantifier* $\exists$;
- (and) as punctuation symbols.

Variables will be normally denoted by the letters u, v, w, x, y, z (possibly subscripted or superscripted) and functions will be normally denoted by the

letters f, g, h (possibly subscripted). Let $TERM$ be the set of *terms* built on FUN and VAR. Terms will be normally denoted by the letters r, s, t (possibly subscripted or superscripted). Given a term t, the set $vars(t) \subseteq VAR$ denotes the set of variables that occur in t. We call *structured term* a term of the form $f(t_1, \ldots, t_m)$, where $m \geq 1$; we call *proper subterm* of t every subterm of t but t. We assume that sequences are contained in $\mathcal{A}$. We denote by $\underline{t}$ a sequence $t_1, \ldots, t_k$ and we write $\underline{t}_{(k)}$ or $\langle t_1, \ldots, t_k \rangle$ if respectively the size or the elements of the sequence are relevant. Moreover we indicate with $\underline{x}\rho$ the sequence of terms obtained applying the substitution ρ to every element of the sequence $\underline{x}$. We call *atom* a predicate of the form $p(t_1, \ldots, t_n)$ where p is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms. When ambiguity does not arise we write $r(t_1, \ldots, t_m)$ as a shorthand for the predicate $r(t_1) \wedge \ldots \wedge r(t_m)$, where r is a predicate symbol of arity 1.

The *truth value of a predicate* $P \in \mathcal{A}$ w.r.t. a substitution α s.t. $vars(P) \subseteq dom(\alpha)$ is defined inductively on the structure of P, and the meaning of an atom is specified as follows:

- $\neg var(t)\alpha$ is true iff $t\alpha \notin VAR$;
- $ground(t)\alpha$ is true iff $vars(t\alpha) = \emptyset$;
- $(t_1 = t_2)\alpha$ is true iff $t_1\alpha = t_2\alpha$ syntactically;
- $(s \preceq t)\alpha$ is true iff $s\alpha$ is a subterm of $t\alpha$;
- $(s \prec t)\alpha$ is true iff $s\alpha$ is a proper subterm of $t\alpha$;
- $invar(s, t)\alpha$ is true iff $vars(s\alpha) \subseteq vars(t\alpha)$;
- $free(x)\alpha$ is true iff $x\alpha \in VAR$ and $x\alpha \notin vars(y\alpha)$ for all $y \in dom(\alpha)$ s.t. $y \neq x$;
- $var(x)\alpha$ is true iff $x\alpha \in VAR$;
- $\neg ground(t)\alpha$ is true iff $vars(t\alpha) \neq \emptyset$;
- $share(s, t)\alpha$ is true iff $vars(s\alpha) \cap vars(t\alpha) \neq \emptyset$;
- $\neg share(s, t)\alpha$ is true iff $vars(s\alpha) \cap vars(t\alpha) = \emptyset$;
- $inst(x, r_1, r_2, y)\alpha$ is true iff $r_1\alpha$ is the sequence $\langle x_1, \ldots, x_m \rangle$, with $x_i \in vars(x\alpha)$ and $x_i \notin vars(y\alpha)$ for $i \in [1, m]$, $r_2\alpha$ is the sequence $\langle t_1, \ldots, t_m \rangle$ and $\{x_1/t_1, \ldots, x_m/t_m\} \in mgu(\{[\![x\alpha, y\alpha]\!]\})$.

Notice that x and y in $inst(x, r_1, r_2, y)$ represent two terms the second of which is an instance of the first. Thus the predicate $inst$ expresses a special case of the unification.
Given two predicates P and Q, we write $P \equiv Q$ to indicate that P and Q are semantically equivalent. We can assume that the predicates $TRUE$ (the predicate true w.r.t. all substitutions) and $FALSE$ (the predicate false w.r.t. all substitutions) are in $\mathcal{A}$, since $TRUE \equiv (var(x) \vee \neg var(x))$ and $FALSE \equiv (var(x) \wedge \neg var(x))$.

Predicates in $\mathcal{A}$ are not in general monotonic, since all atoms built on predicate symbols not in $Pred$ are non-monotonic by definition. So Lemma 2.2 is not sufficient to characterize $sp.\mathcal{U}$: consider for instance the unification $\{[\![x, a]\!]\}$ and

the predicate $var(x)$. Thus a careful analysis of the effect of the unification process on non-monotonic predicates is necessary. The fact that the connective $\neg$ is not in our language guarantees that atoms built on predicate symbols not in *Pred* are the only non-monotonic atoms of the language; this allows a case analysis of the effect of unification on non-monotonic predicates.

We introduce now some assumptions that will be used to simplify the form of the rules for $sp.\mathcal{U}$ that will be introduced in the next section.
Predicates are of the form $\exists \underline{x} P$ where P doesn't contain any quantifier, it is in disjunctive normal form (i.e. it is a disjunction of conjunctions of atoms) and the equalities that occur in each conjunct are expressed by a set of equations in solved form. Atoms with predicate symbol *free*, *var*, $\neg var$, *ground*, $\neg ground$, *share*, $\neg share$, *invar* have variables as arguments. For any formula $sp.\mathcal{U}.P$ the predicate P does not contain (existential) quantifiers.
All assumptions are not restrictive. Here the proof for the last one.

Lemma 2.3 *If the variable x does not occur in $\mathcal{U}$ then $sp.\mathcal{U}.\exists x P$ is equivalent to $\exists x(sp.\mathcal{U}.P)$ w.r.t. Definition 2.1.*

Proof. Since x doesn't occur in $\mathcal{U}$ then the truth value of $\exists x(sp.\mathcal{U}.P)\beta$ and of $(sp.\mathcal{U}.\exists x P)\beta$ does not depend on $x\beta$. Thus we can assume without loss of generality $x \notin dom(\beta)$. Then $(sp.\mathcal{U}.\exists x P)\beta$ is true iff there exist α and μ s.t. $x \notin dom(\alpha)$, $x \notin dom(\mu)$, $\mu \in mgu(\mathcal{U}\alpha)$, $(\exists x P)\alpha$ is true and $\beta = \alpha\mu$ iff there exist α, μ and t s.t. $x \notin dom(\alpha)$, $x \notin dom(\mu)$, $\mu \in mgu(\mathcal{U}\alpha)$, $P(\alpha \cup \{x/t\})$ is true and $\beta = \alpha\mu$ iff there exist α, μ and t s.t. $\mu \in mgu(\mathcal{U}(\alpha \cup \{x/t\}))$, $P(\alpha \cup \{x/t\})$ is true and $(\beta \cup \{x/t\}) = (\alpha \cup \{x/t\})\mu$ iff $(sp.\mathcal{U}.P)(\beta \cup \{x/t\})$ is true iff $(\exists x\, sp.\mathcal{U}.P)\beta$ is true. □

3 A Calculus for $sp.\mathcal{U}$

The following conditions on P and $\mathcal{U}$ characterize the types of formulas which will specify the scope of applicability of the rules for $sp.\mathcal{U}.P$.

(i) P is a conjunction of atoms.
(ii) For each equation $x = t$ in P, x does not occur in $\mathcal{U}$.
(iii) For every x occurring in $\mathcal{U}$ either $var(x)$ or $\neg var(x)$ occurs in P.
(iv) For all distinct variables x occurring in $\mathcal{U}$ and y occurring in P either $share(x, y)$ or $\neg share(x, y)$ occurs in P.
(v) $\mathcal{U} = \{S_1, \ldots, S_n\}$ contains disjoint sets, i.e. $S_i \cap S_j = \emptyset$ for $i \neq j$.
(vi) Each set in $\mathcal{U}$ contains more than one element.
(vii) Each set in $\mathcal{U}$ contains at most one structured element $f(v_1, \ldots, v_m)$ and in such a case $free(v_1), \ldots, free(v_m)$ occur in P.
(viii) Every element x of a set $S \in \mathcal{U}$ is s.t. $free(x)$ occurs in P if x occurs in the structured element of another set in $\mathcal{U}$ and $\neg var(x)$ occurs in P otherwise. Moreover, each set that contains a structured element also contains an element

y s.t. $free(y)$ occurs in P. (Hence y occurs in the structured element of another set).

We introduce 3 types of formulas $sp.\mathcal{U}.P$ as follows.

type 1: those which satisfy conditions (i)–(iii).
type 2: those which satisfy conditions (i)–(vii).
type 3: those which satisfy conditions (i)–(viii).

Each type of formula characterizes a simpler form of P and $\mathcal{U}$. The final form will be a disjunction of formulas in the so called *reduced form.*

A formula $sp.\mathcal{U}.P$ is in **reduced form** if P is a conjunction of atoms, for each equation $x = t$ in P x does not occur in $\mathcal{U}$, $\mathcal{U}$ contains only disjoint sets of two or more variables, for all x occurring in $\mathcal{U}$ both $\neg var(x)$ and $\neg ground(x)$ occur in P and for all x occurring in $\mathcal{U}$ and y occurring in P either $share(x, y)$ or $\neg share(x, y)$ occurs in P.

We are now ready to present the rules for $sp.\mathcal{U}.P$. The notation E_t^x will be used to indicate the formula obtained by replacing the occurrences of x in E with t.

- If $P = P_1 \vee \ldots \vee P_n$ then

$$sp.\mathcal{U}.P \equiv sp.\mathcal{U}.P_1 \vee \ldots \vee sp.\mathcal{U}.P_n \qquad \textbf{OR}$$

- If x occurs in $\mathcal{U}$ and neither $var(x)$ nor $\neg var(x)$ occurs in P then

$$sp.\mathcal{U}.P \equiv sp.\mathcal{U}.(P \wedge var(x)) \vee sp.\mathcal{U}.(P \wedge \neg var(x)) \qquad \textbf{VAR1}$$

- If P is a conjunction of atoms and $x = t$ occurs in P then:

$$sp.\mathcal{U}.P \equiv sp.\mathcal{U}_t^x.P \qquad \textbf{EQ}$$

- $$sp.\mathcal{U}.FALSE \equiv FALSE \qquad \textbf{F}$$

The following eight rules may be applied only to **type 1** formulas.

- If x occurs in $\mathcal{U}$ and y occurs in P and neither $share(x, y)$ nor $\neg share(x, y)$ occurs in P then

$$sp.\mathcal{U}.P \equiv sp.\mathcal{U}.(P \wedge share(x, y)) \vee sp.\mathcal{U}.(P \wedge \neg share(x, y)) \qquad \textbf{SH1}$$

- If $\mathcal{U} = \{[\![f_1(\underline{s}), f_2(\underline{t}), \ldots]\!], S_2, \ldots, S_n\}$ and $f_1 \neq f_2$, then

$$sp.\mathcal{U}.P \equiv FALSE \qquad \textbf{MIS1}$$

- If $\mathcal{U} = \{[\![x, s, \underline{t}]\!], S_2, \ldots, S_n\}$ and either $x \in vars(s)$ or the conjunct $x \prec s$ occurs in P then

$$sp.\mathcal{U}.P \equiv FALSE \qquad \textbf{MIS2}$$

- If $\mathcal{U} = \{[\![f(\underline{s}^1_{(k)}), \ldots, f(\underline{s}^m_{(k)})]\!], S_2, \ldots, S_n\}$ then

$$sp.\mathcal{U}.P \equiv sp.\mathcal{U}'.P \qquad \textbf{STR1}$$

where $\mathcal{U}' = \{[\![\underline{s}^{(m)}_j]\!]_{j\in[1,k]}, S_2, \ldots, S_n\}$

- If $\mathcal{U} = \{[\![f(\underline{s}^1_{(k)}), \ldots, f(\underline{s}^i_{(k)}), x_{i+1}, \ldots, x_m]\!], S_2, \ldots, S_n\}$ with $i < m$ and either $i \geq 2$ or at least one s^i_j is not a variable or at least one s^i_j is a variable such that $\neg var(s^i_j)$ occurs in P, then

$$sp.\mathcal{U}.P \equiv \exists \underline{y}_{(k)} \big(sp.\mathcal{U}'.P'\big) \qquad \textbf{STR2}$$

where $\mathcal{U}' = \{[\![f(\underline{y}_{(k)}), x_{i+1}, \ldots, x_m]\!], [\![y_j, \underline{s}^{(i)}_j]\!]_{j\in[1,k]}, S_2, \ldots, S_n\}$,
$P' = P \wedge free(\underline{y}_{(k)})$ and $\underline{y}_{(k)}$ are fresh variables.

- If $\mathcal{U} = \{[\![t, \underline{t}_{(m)}]\!], [\![t, \underline{s}_{(m')}]\!], S_3, \ldots, S_n\}$ then

$$sp.\mathcal{U}.P \equiv sp.\mathcal{U}'.P \qquad \textbf{SH2}$$

where $\mathcal{U}' = \{[\![t, \underline{t}_{(m)}, \underline{s}_{(m')}]\!], S_3, \ldots, S_n\}$

- If $\mathcal{U} = \{[\![t]\!], S_2, \ldots, S_n\}$ then

$$sp.\mathcal{U}.P \equiv sp.\mathcal{U}'.P \qquad \textbf{SI}$$

where $\mathcal{U}' = \{S_2, \ldots, S_n\}$

The following two rules may be applied only to **type 2** formulas.

- If $\mathcal{U} = \{[\![t, \underline{x}_{(m)}]\!], S_2, \ldots, S_n\}$ where x_m does not occur in the structured term of any set of $\mathcal{U}$, $var(x_m)$ and $\neg share(x_m, y)$ occurs in P for all $y \in vars(t)$, then

$$sp.\mathcal{U}.P \equiv \exists \underline{z}'\ sp.\mathcal{U}'.R \qquad \textbf{VAR2}$$

where $\mathcal{U}' = \{[\![t, \underline{x}_{(m-1)}]\!], S_2, \ldots, S_n\}$,
$R = (\bigwedge_{z\in\underline{z}} inst(z\rho, \langle x_m\rho\rangle, \langle t\rangle, z) \wedge P' \wedge x_m = t)$,
$\underline{z} = \langle z \in vars(P) \mid P \Rightarrow share(z, x_m)\rangle$, $\underline{z}' = \underline{z}\rho$ is a variant of $\underline{z}$ disjoint from P and $P' = P^{\underline{z}}_{\underline{z}'}$.

- If $\mathcal{U} = \{[\![f(\underline{s}_{(k)}), \underline{x}_{(m)}]\!], S_2, \ldots, S_n\}$ and $\neg var(x_1), \ldots, \neg var(x_m)$ occur in P then

$$sp.\mathcal{U}.P \equiv \exists \underline{y}\ sp.\mathcal{U}'.(P \wedge x_1 = f(\underline{y}^1_{(k)}) \wedge \ldots \wedge x_m = f(\underline{y}^m_{(k)})) \qquad \textbf{VAR3}$$

where $\mathcal{U}' = \{[\![s_i, \underline{y}^{(m)}_i]\!]_{i\in[1,k]}, S_2, \ldots, S_n\}$ and $\underline{y}$ is the sequence $\underline{y}^1_{(k)}, \ldots, \underline{y}^m_{(k)}$ of fresh variables.

The following three rules may be applied only to **type 3** formulas.

- If there is a set $S \in \mathcal{U}$ that contains a structured term then

$$sp\mathcal{U}.P \equiv FALSE \qquad \textbf{MIS3}$$

- If x occurs in $\mathcal{U}$ and neither $ground(x)$ nor $\neg ground(x)$ occurs in P then

$$sp\mathcal{U}.P \equiv sp\mathcal{U}.(P \wedge ground(x)) \vee sp\mathcal{U}.(P \wedge \neg ground(x)) \qquad \textbf{GR1}$$

- If $\mathcal{U} = \{[\![\underline{x}_{(m)}]\!], S_2, \ldots, S_n\}$ and $ground(x_m)$ occurs in P then

$$sp\mathcal{U}.P \equiv \exists \underline{x}', z_{\underline{x}}, y_{\underline{x}}\ sp\mathcal{U}'.R \qquad \textbf{GR2}$$

where $\mathcal{U}' = \{S_2, \ldots, S_n\}$,
$\underline{x} = \langle x \in vars(P) \mid P \Rightarrow share(x, x_i)$ for some $i \in [1, m-1]\rangle$, $\underline{x}' = \underline{x}\rho$ is a variant of $\underline{x}$ disjoint from P, $z_{\underline{x}}$ and $y_{\underline{x}}$ are the sequences of fresh variables z_x and y_x with $x \in \underline{x}$, $P' = P^{\underline{x}}_{\underline{x}'}$ and R is the predicate

$\bigwedge_{x \in \underline{x}}(inst(x\rho, z_x, y_x, x) \bigwedge_{i \mid P \Rightarrow share(x, x_i)} invar(z_x, z_{x_i}) \bigwedge_{y \in vars(P)} \neg share(z_x, y))$
$\wedge P' \wedge x_1 = \ldots = x_m.$

To a formula in **reduced form** we can apply the following rule.

- If $sp\mathcal{U}.P$ is in reduced form, where $\mathcal{U} = \{[\![\underline{x}^1_{(m_1)}]\!], \ldots, [\![\underline{x}^n_{(m_n)}]\!]\}$, then

$$sp\mathcal{U}.P \equiv \exists \underline{x}', z_{\underline{x}}, y_{\underline{x}}\ (R \wedge U) \qquad \textbf{RF}$$

where U is the predicate $(x^1_1 = \ldots = x^1_{m_1}) \wedge \ldots \wedge (x^n_1 = \ldots = x^n_{m_n})$,
$\underline{x} = \langle x \in vars(P) \mid P \Rightarrow share(x, x^j_i)$ for some $i \in [1, m_j], j \in [1, n]\rangle$, $\underline{x}' = \underline{x}\rho$ is a variant of $\underline{x}$ disjoint from P, $z_{\underline{x}}$ and $y_{\underline{x}}$ are the sequences of fresh variables z_x and y_x with $x \in \underline{x}$, $P' = P^{\underline{x}}_{\underline{x}'}$ and $R = (\bigwedge_{x \in \underline{x}} inst(x\rho, z_x, y_x, x) \wedge P')$.

The previous rules are natural abstractions of the relative unification step except rules **MIS3**,**VAR2**, **GR2** and **RF**. Rule **MIS3** relies on the condition that the formula is of type 3 and $\mathcal{U}$ contains at least a set with a structured element. In this case it can be proven that $\mathcal{U}$ has no unifier.
Rules **VAR2**, **GR2** and **RF** take into account how sharing among variables can propagate the bindings produced by the considered transformation and how the transformations affect the truth of the non-monotonic atoms. To keep track of the way the predicate is modified suitable variables are renamed with fresh variables existentially quantified and suitable predicates are introduced to specify the link among the original variables and the renamed ones.
All the rules are syntactic. Thus the set of rules provides a (nondeterministic) algorithm. We will see in the following section that this algorithm terminates and computes $sp\mathcal{U}.P$. We conclude this section with some examples.

Let $P = free(x, y)$ and $\mathcal{U} = \{[\![f(x), y]\!], [\![g(y), x]\!]\}$. Since $sp\mathcal{U}.P$ is of **type 3**, then by rule **MIS3** it is equivalent to $FALSE$. In fact an occur check does occur.

Let $P = (free(x,y) \wedge \neg share(x,y))$ and $\mathcal{U} = \{[\![f(y),x]\!]\}$. Since $sp.\mathcal{U}.P$ is of **type 2**, then we can apply rule **VAR2**. We obtain

$\exists x'(sp.\{[\![f(y)]\!]\}.(P^{x}_{x'} \wedge inst(x', \langle x'\rangle, \langle f(y)\rangle, x) \wedge x = f(y)))$.

By rules **SI** and **RF** we obtain

$\exists x'(P^{x}_{x'} \wedge inst(x', \langle x'\rangle, \langle f(y)\rangle, x) \wedge x = f(y))$,

which is equivalent to $(free(y) \wedge x = f(y))$.

Let $P = (ground(y) \wedge \neg var(x) \wedge \neg ground(x) \wedge \neg share(x,y))$ and $\mathcal{U} = \{[\![x,y]\!]\}$. Since $sp.\mathcal{U}.P$ is of **type 3**, then we can apply rule **GR2**. We obtain

$\exists x', z_x, y_x(sp.\{\}.$
$(P^{x}_{x'} \wedge inst(x', z_x, y_x, x) \wedge invar(z_x, z_x) \wedge \neg share(z_x, x) \wedge \neg share(z_x, y) \wedge x = y))$.

By rule **RF** we obtain

$\exists x', z_x, y_x(P^{x}_{x'} \wedge inst(x', z_x, y_x, x) \wedge invar(z_x, z_x) \wedge$
$\neg share(z_x, x) \wedge \neg share(z_x, y) \wedge x = y)$

which is equivalent to $(ground(y) \wedge \neg var(y) \wedge y = x)$.

Let $P = (\neg var(x,y) \wedge \neg ground(x,y) \wedge share(x,y))$ and $\mathcal{U} = \{[\![x,y]\!]\}$. Since $sp.\mathcal{U}.P$ is in **reduced form** , then we can apply rule **RF**. We obtain

$\exists x', y', z_x, y_x, z_y, y_y(P^{x,y}_{x',y'} \wedge inst(x', z_x, y_x, x) \wedge$
$inst(x', z_x, y_x, y) \wedge inst(y', z_y, y_y, y) \wedge x = y))$,

which is equivalent to $(x = y \wedge \neg var(x,y))$, if CON contains at least a function of arity greater than one and a constant; otherwise it is equivalent to $(x = y \wedge \neg var(x,y) \wedge \neg ground(x,y))$.

4 Soundness and Completeness of the Calculus

We indicate by $\mathcal{H}_{sp}$ the set of rules but **RF**. We first show that all the rules are equivalences. Then we show that a formula $sp.\mathcal{U}.P$ can be reduced in a finite number of steps to a disjunction of formulas in reduced form, by applying rules from $\mathcal{H}_{sp}$. Finally rule **RF** applied to each disjunct will give the desired predicate (of $\mathcal{A}$) relative to $sp.\mathcal{U}.P$.

Theorem 4.1 *All rules are equivalences (with respect to Definition 2.1)*

Proof. The proof is not difficult except for rules **MIS3**, **VAR2**, **GR2** and **RF** which have a quite technical proof.

MIS3 By hypothesis the formula is of type 3 and $\mathcal{U}$ contains at least a set with a structured element. Then by condition (vii) each set that contains a structured element $f(y_1, \ldots, y_k)$ also contains at least a variable x that occurs in the structured element of another set. In such a situation we can eventually extract from $\mathcal{U}$ a subset $\{S_1, \ldots, S_t\}$ of sets such that

$S_1 = [\![f_1(\ldots, x_t, \ldots), x_1, \ldots]\!]$
$S_2 = [\![f_2(\ldots, x_1, \ldots), x_2, \ldots]\!]$

...
$S_t = [\![f_t(\ldots, x_{t-1}, \ldots), x_t, \ldots]\!]$.
Clearly $\{S_1, \ldots, S_t\}\alpha$ has no unifier.

In the next proofs we use the following properties of most general unifiers:
1) Let $\mathcal{U} = \{[\![\underline{t}_{(m)}]\!], S_2, \ldots S_n\}$. If $\beta \in mgu(\{[\![\underline{t}_{(i)}]\!]\})$ and $\mu \in mgu(\mathcal{U}\beta)$ then $\mu \cup \mu' \in mgu(\mathcal{U})$, where $\mu' = (\beta\mu)_{|dom(\beta)}$.
2) $mgu(\{[\![t]\!], S_2, \ldots S_n\}) = mgu(\{S_2, \ldots S_n\})$.

VAR2 Let α be such that $P\alpha$ is true and let $\mu \in mgu(\mathcal{U}\alpha)$. Let α' be such that

$$x\alpha' = \begin{cases} t\alpha & \text{if } x = x_m, \\ z\alpha & \text{if } x = z\rho, \\ (x\alpha)^{x_m\alpha}_{t\alpha} & \text{otherwise.} \end{cases}$$

Let A' be an atom in P'. Then $A' = A^{\underline{z}}_{\underline{z}'}$ with A atom in P. If A' is monotonic then $A'\alpha'$ is an instance of $A\alpha$. Otherwise $A'\alpha' = A\alpha$. Thus in both cases $A'\alpha'$ is true. From $t\alpha' = t\alpha$ it follows that $inst(z\rho, \langle x_m\rho\rangle, \langle t\rangle, z)\alpha'$ and $(x_m = t)\alpha'$ are both true. Then $R\alpha'$ is true. Now let μ' be s.t. $\mu = \mu' \cup \{x_m\alpha/t\alpha\mu'\}$. Then from $\mathcal{U}'\alpha' = (\mathcal{U}\alpha)^{x_m\alpha}_{t\alpha}$ it follows by property 1) that $\mu' \in mgu(\mathcal{U}'\alpha')$. Thus $(sp.\mathcal{U}'.R)\alpha'\mu'$ is true and, since $x\alpha\mu = x\alpha'\mu'$ for all x in P, then $(\exists \underline{z}'\ sp.\mathcal{U}'.R)\alpha\mu$ is true.
Viceversa let α' be such that $R\alpha'$ is true and let $\mu' \in mgu(\mathcal{U}'\alpha')$. Let α be such that

$$x\alpha = \begin{cases} x\rho\alpha' & \text{if } x \text{ in } \underline{z}, \\ x\alpha' & \text{otherwise.} \end{cases}$$

Then $P\alpha = P'\alpha'$ is true. Let $\mu = \mu' \cup \{x_m\alpha/t\alpha\mu'\}$. By $inst(z\rho, \langle x_m\rho\rangle, \langle t\rangle, z)\alpha'$ true for all z in $\underline{z}$ it follows that $\mathcal{U}'\alpha' = (\mathcal{U}\alpha)^{x_m\alpha}_{t\alpha}$. Then by property 1) $\mu \in mgu(\mathcal{U}\alpha)$. Thus $(sp.\mathcal{U}.P)\alpha\mu$ is true and, since $x_m\alpha'\mu' = t\alpha'\mu' = t\alpha\mu' = x_m\alpha\mu$, then $(sp.\mathcal{U}.P)\alpha'\mu'$ is true.

GR2 Let α and μ be such that $P\alpha$ is true and $\mu \in mgu(\mathcal{U}\alpha)$, let $\mu_i = \mu_{|vars(x_i\alpha)}$. From $ground(x_m)\alpha$ true it follows that $x_i\alpha\mu = x_m\alpha$ for $i \in [1, m-1]$. Let α' be s.t.

$$w\alpha' = \begin{cases} x\alpha & \text{if } w = x\rho \text{ with } x \text{ in } \underline{x}, \\ x_m\alpha & \text{if } w = x_i \text{ for } i \in [1, m_1], \\ \underline{y}^1 \ldots \underline{y}^{m-1} & \text{if } w = z_x \text{ with } x \text{ in } \underline{x}, \\ (\underline{y}^1 \ldots \underline{y}^{m-1})\mu & \text{if } w = y_x \text{ with } x \text{ in } \underline{x}, \\ (w\alpha)\mu_1 \ldots \mu_{m-1} & \text{otherwise.} \end{cases}$$

where $\underline{y}^i$ is the sequence of variables in $vars(x\alpha) \cap vars(x_i\alpha)$ for $i \in [1, m-1]$. Let A' be an atom of P'. Then $A' = A^{\underline{x}}_{\underline{x}'}$ with A atom in P. If A' is monotonic then $A'\alpha'$ is an instance of $A\alpha$. If A' is non-monotonic then $A'\alpha' = A\alpha$. In both cases $A'\alpha'$ is true. Moreover $(x_1 = \ldots = x_m)\alpha'$ is true because $x_m\alpha' = x_m\alpha = x_i\alpha'$ for all $i \in [1, m-1]$, $\neg share(z_x, x)\alpha'$ is true because all variables in $z_x\alpha'$ occur in $x_i\alpha$ for some $i \in [1, m-1]$ and $x\alpha'$ is obtained replacing the variables in

all $x_i\alpha$ with ground terms, $inst(x\rho, z_x, y_x, x)\alpha'$ is true because $x\rho\alpha' = x\alpha$, $x\alpha' = (x\alpha)^{z_x\alpha'}_{y_x\alpha'}$ and $\neg share(z_x, x)\alpha'$ true imply $\{z_x\alpha'/y_x\alpha'\} \in mgu(\{[\![x\rho\alpha', x\alpha']\!]\})$; finally $invar(z_x, \langle z_{x_1}, \ldots, z_{x_{m-1}}\rangle)\alpha'$ is true by construction. Thus $R\alpha'$ is true. Now let $\mu' = \mu_{|vars(\mathcal{U}'\alpha')}$. We have that $\mu_1 \ldots \mu_{m-1}$ is in $mgu(\{[\![x_1, \ldots, x_m]\!]\}\alpha)$, $range(\mu_1 \ldots \mu_{m-1}) = \emptyset$ because $x_m\alpha$ is ground, $\mathcal{U}'\alpha' = \mathcal{U}'\alpha\mu_1 \ldots \mu_{m-1}$. Then $\mu = \mu' \cup \mu_1 \ldots \mu_{m-1}$ and by properties 1) and 2) μ' is in $mgu(\mathcal{U}'\alpha')$. Then $(sp.\mathcal{U}'.R)\alpha'\mu'$ is true and, since $x\alpha'\mu' = x\alpha\mu$ for all x occurring in P, then $(\exists \underline{x}, z_{\underline{x}}, y_{\underline{x}}\ sp.\mathcal{U}.R)\alpha\mu$ is true.
Viceversa let α' be such that $R\alpha'$ is true and let $\mu' \in mgu(\mathcal{U}'\alpha')$. Let α be s.t.

$$x\alpha = \begin{cases} x\rho\alpha' & \text{if } x \text{ in } \underline{x}, \\ x\alpha' & \text{otherwise.} \end{cases}$$

Then $P\alpha = P'\alpha'$ is true. Let $\mu = \mu' \cup \beta$ with $\beta = \{(z_{x_i}\alpha'/y_{x_i}\alpha')_{i\in[1,m-1]}\}$. From $inst(x\rho, z_x, y_x, x)\alpha'$, $\neg share(z_{x_j}, x)\alpha'$ and $invar(z_x, \langle z_{x_1}, \ldots, z_{x_{m-1}}\rangle)\alpha'$ true it follows that $x\alpha' = x\alpha\beta$ for all $x \in \underline{x}$. If $x \notin \underline{x}$ then from $\neg share(z_y, x)\alpha'$ true for all y it follows that $x\alpha' = (x\alpha')\beta = x\alpha\beta$. Then $x\alpha' = x\alpha\beta$ for all x occurring in P. Then $\mathcal{U}'\alpha' = \mathcal{U}'\alpha\beta$. From $x_1\alpha' = \ldots = x_m\alpha'$ true, $x_m\alpha'$ ground and $inst(x_i\rho, z_{x_i}, y_{x_i}, x_i)\alpha'$ true for all $i \in [1, m-1]$ it follows that $\beta \in mgu(\{[\![x_1, \ldots, x_m]\!]\}\alpha)$. Then by properties 1) and 2) it follows that $\mu \in mgu(\mathcal{U}\alpha)$. Thus $(sp.\mathcal{U}.P)\alpha\mu$ is true and, since $x\alpha'\mu' = (x\alpha)\beta\mu' = x\alpha\mu$ for all x occurring in P, then $(sp.\mathcal{U}.P)\alpha'\mu'$ is true.

RF Let α and μ be such that $P\alpha$ is true, $\mu \in mgu(\mathcal{U}\alpha)$. Let α' be s.t.

$$w\alpha' = \begin{cases} x\alpha & \text{if } w = x\rho \text{ with } x \text{ in } \underline{x}, \\ w\alpha\mu & \text{if } w \text{ occurs in } P, \\ \underline{y} & \text{if } w = z_x \text{ with } x \text{ in } \underline{x}, \\ \underline{y}\mu & \text{if } w = y_x \text{ with } x \text{ in } \underline{x}. \end{cases}$$

where $\underline{y}$ is the sequence of variables occurring in $dom(\mu_{|vars(x\alpha)})$. Now $U\alpha'$ is true because $x_i^j\alpha' = x_i^j\alpha\mu$ for every $i \in [1, m_j]$, $j \in [1, n]$. Let A' be an atom of P'. Then $A' = A^{\underline{x}}_{\underline{x}'}$ with A atom in P. If A' is monotonic then $A'\alpha'$ is an instance of $A\alpha$. If A' is non-monotonic then $A'\alpha' = A\alpha$. In both cases $A'\alpha'$ is true. Moreover $inst(x\rho, z_x, y_x, x)\alpha'$ is true because $x\rho\alpha' = x\alpha$, $x\alpha' = x\alpha\mu$ and the substitution relative to the two sequences $z_x\alpha'$ and $y_x\alpha'$ is equal to $\mu_{|vars(x\alpha)}$. Since μ is idempotent by hypothesis, then $\mu_{|vars(x\alpha)} \in mgu(\{[\![x\alpha, x\alpha\mu]\!]\})$. Then $(R \wedge U)\alpha'$ is true and, since $x\alpha' = x\alpha\mu$ for every x occurring in P, then $(\exists \underline{x}', z_{\underline{x}}, y_{\underline{x}}(R \wedge U))\alpha\mu$ is true.
Viceversa let α' be s.t. $(R \wedge U)\alpha'$ is true. Let α be s.t.

$$x\alpha = \begin{cases} x\rho\alpha' & \text{if } x \text{ in } \underline{x}, \\ x\alpha' & \text{otherwise.} \end{cases}$$

Then $P\alpha = P'\alpha'$ is true. Let μ be the substitution relative to the sequences $z_{x_i^j}\alpha'$, $y_{x_i^j}\alpha'$ for all $i \in [1, m_j]$, $j \in [1, n]$. Then $\mu \in mgu(\mathcal{U}\alpha)$. Thus $(sp.\mathcal{U}.P)\alpha\mu$

is true and, since $x\alpha\mu = x\rho\alpha'\mu = x\alpha'$ for every x that occurs in P, then $(sp\mathcal{U}.P)\alpha'$ is true. □

Theorem 4.2 *The system $\mathcal{H}_{sp}$ is terminating.*

Proof. (Sketch)

We show that no proof tree built using $\mathcal{H}_{sp}$ has an infinite branch. Rules **F**, **MIS1**, **MIS2** and **MIS3** have a predicate as right hand side, so they cannot belong to an infinite branch. To prove that only finitely many applications of the remaining rules are allowed, consider the tuple

$\tau = (leq, comp, funct, elem, disj, unvar, unshare, unground)$

of natural numbers with the lexicographic order. A structured term $f(t_1, \ldots, t_n)$ will be called *compound* if either some t_i is not a variable or the variables $t_1, \ldots, t_n$ are not distinct. Then *leq* denotes the number of variables in $\mathcal{U}$ that occur as left hand side of an equation in P, *comp* denotes the number of occurences of compound subterms of terms in $\mathcal{U}$, *funct* denotes the number of occurences of functor symbols in $\mathcal{U}$, *elem* denotes the total number of elements in the sets of $\mathcal{U}$, *disj* denotes the number of disjuncts in the disjunctive normal form of P, *unvar* denotes the number of variables x in P such that neither $P \Rightarrow var(x)$ nor $P \Rightarrow \neg var(x)$ holds, *unshare* denotes the number of variables x in P such that neither $P \Rightarrow share(x, y)$ nor $P \Rightarrow \neg share(x, y)$ holds for some variable y distinct by x, *unground* denotes the number of variables x in P such that neither $P \Rightarrow ground(x)$ nor $P \Rightarrow \neg ground(x)$ holds.

It is not difficult to check that the application of every rule of $\mathcal{H}_{sp}$ decreases the value of τ. □

Corollary 4.3 *Rules of $\mathcal{H}_{sp}$ transform $sp\mathcal{U}.P$ in a (semantically unique) disjunction of formulas in reduced form.*

Proof. (Sketch)

By Theorem 4.1 all transformations are equivalences (w.r.t. Definition 2.1). By Theorem 4.2 there is a final form. Thus the final form is semantically unique. By contrapposition it is not difficult to show that if the final form is not a disjunction of formulas in reduced form then one of the rules in $\mathcal{H}_{sp}$ may be applied. □

5 Applications

Predicate transformers are related to the core of abstract interpretation of imperative programs. In [2] predicate transformers are used to define deductive semantics. Deductive semantics is used to design approximate program analysis frameworks. To propose a similar approach in the setting of logic programming we need the correspondent of program point for a logic program. In [7] Nilsson introduced a scheme for inferring run-time properties of logic programs based on

a semantic description of logic programs that uses the concept of program point. The predicate transformer *sp* can be easily cast in such a theory. A clause of a logic program $\mathcal{P}$ is interpreted as a sequence of procedure calls. To each call A there corresponds a *calling point* ${}_\bullet A$ and a *success point* $A_\bullet$. The leftmost and rightmost points in the body of a clause C are called respectively *entry-* and *exit points* of the clause and are indicated respectively by ${}_\bullet C$ and $C_\bullet$. Goals are represented as elements of the set $\mathcal{C}goals := (\mathcal{P}oints \times \mathcal{E}nv)^*$, where $\mathcal{P}oints$ denotes the set of program points of $\mathcal{P}$ and $\mathcal{E}nv$ is the set of predicates $\mathcal{A}$. A transition system for $\mathcal{P}$ can be defined through two state transition schemes that transform elements of $\mathcal{C}goals$ as follows.

$$\langle C_\bullet; R\rangle :: y \models y,$$

$$\langle {}_\bullet A; R\rangle :: y \models (\langle {}_\bullet C; TRUE\rangle :: \langle A_\bullet; R\rangle :: y\rangle)(T\sigma^{-1}),$$

where A is a body atom, $C\sigma$ is a variant of a clause C of $\mathcal{P}$ s.t. $vars(\langle {}_\bullet A; R\rangle :: y) \cap vars(C\sigma) = \emptyset$, $T \equiv sp.\{[\![A, head(C\sigma)]\!]\}.(R \wedge free(vars(C\sigma))) \not\equiv FALSE$. We assume that the program clauses are disjoint and that the definition of $\mathcal{U}$ in *sp* is generalized in the obvious way to atoms or terms. The application of a predicate R to a $\mathcal{C}$-goal is defined as follows:
$(nil)R = R$,
$(\langle x; T\rangle :: y)R = \langle x; T \bullet R\rangle :: yR$,
where $T \bullet R$ is (equivalent to) $T' \wedge R$, with T' the strongest assertion (w.r.t. implication) s.t. $T \rightarrow T'$ and $(T' \wedge R) \not\equiv FALSE$. Notice that $T \bullet R$ is defined when R is consistent. For instance if $T = (x = y \wedge var(x))$ and $R = ground(y)$ then $T \bullet R \equiv (x = y \wedge ground(y))$.

The previous transitions schemes are obtained from those in [7] by taking as enviroment $\mathcal{E}nv$ predicates instead of substitutions, by using the predicate transformer *sp* instead of the mgu as operation in the transition and the operation $\bullet$ to model the application of a predicate to a $\mathcal{C}-$goal.
To each program point i is associated a set Θ_i of states which specifies when the program point becomes current. The set of states is defined as $\mathcal{C}goals \times \mathcal{C}goals$, where the first component describes the $\mathcal{C}$-goal that invoked the clause containing point i and the second component is the $\mathcal{C}$-goal when the point is current. The semantics of $\mathcal{P}$ is defined as the least fixpoint of the system of equations relative to its program points. Every program point is either the entry point of a clause or the success point of a body atom. Then it is sufficient to define the meaning of entry- and success points:

$$\Theta_{\bullet C} = \bigcup_{A \rightsquigarrow C} \{\langle G_i; G_{i+1}\rangle \mid \exists G(\langle G; G_i\rangle \in \Theta_{\bullet A} \wedge G_i \models^C G_{i+1})\},$$

$$\Theta_{A_\bullet} = \bigcup_{A \rightsquigarrow C} \{\langle G; tail(G_j)\rangle \mid \exists G_i(\langle G; G_i\rangle \in \Theta_{\bullet A} \wedge \langle G_i; G_j\rangle \in \Theta_{C_\bullet})\}.$$

Example Consider the following simple case of concatenation of two lists:

$C_0 : \leftarrow_1 append([a], [], z)_2.$
$C_1 : append([H|L1], L2, [H|L3]) \leftarrow_3 append(L1, L2, L3)_4.$

C_2 : $append([\,], L, L) \leftarrow_5$.

Here the program points are explicitely labelled by integers. The meaning of this program, when $append([a], [\,], z)$ is called with z $free$ variable, can be given as least fixpoint of the following set of equations, where we use the notation of [7].

$\Theta_1 = \{\langle nil\ ;\ \langle_\bullet C_0; free(z)\rangle :: nil\rangle\}$,
$\Theta_2 = \{\langle G; tail(G_j)\rangle \mid \exists G_i(\langle G; G_i\rangle \in \Theta_1 \wedge (\langle G_i; G_j\rangle \in \Theta_4 \vee \langle G_i; G_j\rangle \in \Theta_5))\}$,
$\Theta_3 = \{\langle G_i; G_{i+1}\rangle \mid \exists G((\langle G; G_i\rangle \in \Theta_1 \vee \langle G; G_i\rangle \in \Theta_3) \wedge G_i \models^{C_1} G_{i+1})\}$,
$\Theta_4 = \{\langle G; tail(G_j)\rangle \mid \exists G_i(\langle G; G_i\rangle \in \Theta_3 \wedge (\langle G_i; G_j\rangle \in \Theta_4 \vee \langle G_i; G_j\rangle \in \Theta_5))\}$,
$\Theta_5 = \{\langle G_i; G_{i+1}\rangle \mid \exists G((\langle G; G_i\rangle \in \Theta_1 \vee \langle G; G_i\rangle \in \Theta_3) \wedge G_i \models^{C_2} G_{i+1})\}$.

Notice that in this case the fixpoint can be computed in finite time since the program terminates. We first calculate Θ_3. We need to compute

$sp.\{[\![append([a], [\,], z), append([H|L1], L2, [H|L3])]\!]\}.(free(z, H, L1, L2, L3))$.

By rule **STR1**, rule **VAR2** applied to $L2$ and z, rules **SI**, **STR1** and rule **VAR2** applied to H and $L1$ we obtain the predicate

$T \equiv (H = a \wedge L1 = [\,] \wedge L2 = [\,] \wedge z = [a|L3] \wedge free(L3))$.

Since $(free(z) \bullet T) = T$ then $\langle_\bullet C_0; free(z)\rangle :: nil \models^{C_1} \langle_\bullet C_1; T\rangle :: \langle C_{0\bullet}; T\rangle :: nil$.

By rules **STR1** and **MIS1**
$sp.\{[\![append(L1, L2, L3), append([H'|L1'], L2', [H'|L3'])]\!]\}.$
$(free(H', L1', L2', L3') \wedge T)$
is equivalent to $FALSE$. Hence

$\Theta_3 = \{\langle\langle_\bullet C_0; free(z)\rangle :: nil; \langle_\bullet C_1; T\rangle :: \langle C_{0\bullet}; T\rangle :: nil\rangle\}$.

Consider now Θ_5. We need to compute

$sp.\{[\![append([\,], L, L), append(L1, L2, L3)]\!]\}.(free(L) \wedge T)$.

By rule **STR1**, rule **EQ** applied to $L1$ and $L2$, rule **SI**, rule **SH2** applied to L and rule **VAR2** applied to L and $L3$ we obtain the predicate

$R \equiv ((H = a \wedge L1 = L2 = L3 = L = [\,] \wedge z = [a])$.

Since $T \bullet R = R$ then

$\langle_\bullet C_1; T\rangle :: \langle C_{0\bullet}; T\rangle :: nil \models^{C_2} \langle_\bullet C_2; R\rangle :: \langle C_{1\bullet}; R\rangle :: \langle C_{0\bullet}; R\rangle :: nil$.

By rules **STR1** and **MIS1**
$sp.\{[\![append([a], [\,], z), append([\,], L, L)]\!]\}.(free(L) \wedge T)$
is equivalent to $FALSE$. Hence

$\Theta_5 = \{\langle\langle_\bullet C_1; T\rangle :: \langle C_{0\bullet}; T\rangle :: nil; \langle_\bullet C_2; R\rangle :: \langle C_{1\bullet}; R\rangle :: \langle C_{0\bullet}; R\rangle :: nil\rangle\}$.

Finally Θ_2 and Θ_4 can be easily calculated.

$\Theta_2 = \{\langle nil; \langle C_{0\bullet}; R\rangle :: nil\rangle\};$

$\Theta_4 = \{\langle\langle_\bullet C_0; free(z)\rangle :: nil; \langle C_{1\bullet}; R\rangle :: \langle C_{0\bullet}; R\rangle :: nil\rangle\}.$

Every set Θ_i describes the states associated to the program point i. Thus for instance Θ_3 specifies that the program point 3 becomes current only when the goal $append([a], [\,], z)$ invokes C_1 with z free variable and in such a case H becomes equal to a, $L1$ and $L2$ become equal to the empty list $[\,]$ and $L3$ remains a free variable.

Acknowledgements This research was supported by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of CNR under the grant n. 89.00026.69.

References

[1] P. Cousot, R. Cousot. Abstract Interpretation : a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*, 238–251, 1977.

[2] P. Cousot, R. Cousot. Systematic Design of Program Analysis Frameworks. *Proceedings of the 6th ACM Symposium on Principles of Programming Languages*, 269–282, 1979.

[3] E. W. Dijkstra. A Discipline of Programming. *Prentice-Hall*, 1976.

[4] E. Eder. Properties of Substitutions and Unifications. *Journal of Symbolic Computation*, 1: 31–46, 1985.

[5] R.W. Floyd. Assigning Meanings to Programs. *Proc. Symp. Appl. Math., American Math. Society, Providence, Rhode Island*, 19: 15–32, 1967.

[6] J-L. Lassez, M.J. Maher, K. Marriott. Unification revisited. *Fundations of Logic and Functional Programming, LNCS 306*, 1987.

[7] Ulf Nilsson. Systematic Semantics Approximations of Logic Programs. *Proceedings of PLILP '90, Springer-Verlag*, 1990.