# Semantic models for concurrent logic languages*

F.S. de Boer and J.J.M.M. Rutten

*Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, Netherlands*

J.N. Kok

*Department of Computer Science, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, Netherlands*

C. Palamidessi

*Dipartimento di Informatica, Universitá di Pisa, Corso Italia 40, 56100 Pisa, Italy*

*Abstract*

de Boer, F.S., J.J.M.M. Rutten, J.N. Kok and C. Palamidessi, Semantic models for concurrent logic languages, Theoretical Computer Science 86 (1991) 3-33.

In this paper we develop semantic models for a class of concurrent logic languages. We give two operational semantics based on a transition system, a declarative semantics and a denotational semantics. One operational and the declarative semantics model the success set, that is, the set of computed answer substitutions corresponding to all successfully terminating computations. The other operational and the denotational semantics also model deadlock and infinite computations. For the declarative and the denotational semantics we extend standard notions such as unification in order to cope with the synchronization mechanism of the class of languages we study. The basic mathematical structure for the declarative semantics is the complete lattice of sets of finite streams of substitutions. In the denotational semantics, we use a complete metric space of tree-like structures that are labelled with functions that represent the basic unification step. We look at the relations between the different models. We relate first the two operational semantics and next the declarative and denotational semantics with their respective operational counterparts.

## 1. Introduction

We study the semantics of a general paradigm for concurrent logic programming languages. It consists of Horn Clause Logic (HCL), to which the following programming mechanisms are added:

- the *input-constraints*, on which the mechanism of synchronization between AND-processes is based, and

* Part of this work was carried out in the context of ESPRIT 415: Parallel Architectures and Languages for Advanced Information Processing — a VLSI-directed approach.

- the *commit* operator, which realizes the so-called *don't care nondeterminism*, controlled by guards.

Examples of languages in this class are PARLOG [4, 5, 13], Guarded Horn Clauses [28, 29], Concurrent Prolog [25, 26], and their so-called flat versions. They can be obtained by specializing the definitions of constraints and commit, and (in some cases) by adding some extra programming primitives. Whereas the addition of constraints and commit leads to a class of very expressive programming languages, the price to be paid is the loss of the clean declarative (model theoretic) understanding of HCL (cf. [27]). Its semantics is affected in a number of respects:

- the *success set* is reduced by the input-constraints,
- the *finite failure set* is enlarged by the commit, and modified (i.e., either reduced or enlarged) by the input-constraints,
- the *infinite failure set* is modified both by the commit and the input-constraints.

Although many operational models have already been investigated [22, 23, 24, 3, 6], a satisfactory declarative one is still to be defined. In this paper we address the problem of characterizing these success and failure sets, first in a declarative and then in a compositional way (i.e., by giving the meaning of a composite goal in terms of the meaning of its conjuncts).

As a running example, we take a language that could be seen as a version of PARLOG. It contains the commit operator and has as input constraints the so-called *declaration modes* of PARLOG.

First, we describe the success and failure sets of this language by a formal operational semantics based on a transition relation (in the style of [14]; see also [23, 6] for similar approaches). The operational meaning is given in terms of sets of *words* (or *streams*) of substitutions, that correspond to the answers computed during the derivation.

Next, we characterize declaratively the new success set as the least fixed-point of an *immediate consequence* operator on interpretations. Here the term declarative indicates that this semantics models computations in a bottom-up fashion. (A model-theoretic semantics is still under investigation.) Our approach can be considered an evolution of the one developed in [18] and [19]. The basis idea there is to model the ability of a process to produce and to consume data structures. This is done by introducing annotations on data structures (terms) and by extending the Herbrand universe with variables (see also [11, 12]). However, the declarative semantics presented in those papers is not able to fully characterize the behaviour of concurrent logic languages. The main problem has to do with the situation of *deadlock* that arises when two processes are obliged to wait for each other for bindings. Consider, for example, the goal $\leftarrow p(x, y), q(x, y)$ and the programs

$$P_1 = \{p(a, b) \leftarrow |., q(a, b) \leftarrow |.\},$$

$$P_2 = \{p(z, b) \leftarrow | r(z)., r(a) \leftarrow |., q(a, b) \leftarrow |.\},$$

and assume that in both cases the first argument of $p$ and the second argument of $q$ are input-constrained (expressed in PARLOG by the declaration modes $D_1 =$

$\{p(?, \hat{}), q(\hat{}, ?)\}$, and $D_2 = \{p(?, \hat{}), q(\hat{}, ?), r(?)\}$, respectively). According to the operational semantics, the computation of the goal cannot succeed in $P_1$ (it results in a deadlock), whilst in $P_2$ it always can. Now it is the case that the approach presented in [18] and [19] is not able to distinguish between the two situations. Indeed, both $p(a^+, b^+)$ and $q(a^+, b^+)$ ($p$ and $q$ producing $a$ and $b$) happen there to be *true* in the models (and in the least fixed-point interpretation) of both the programs. So, a full completeness result (between the declarative and the operational semantics) could not be obtained. For a detailed discussion of this problem see also [16].

Our solution to this problem consists of enriching the interpretations with streams of substitutions. Due to the presence of guards, whose evaluation has to be interpreted as an internal action, the streams of the operational semantics offer too little structure, and we have to add some delimiters to represent *critical sections.* We call these new structures *sequences.* This allows us to characterize declaratively (and, therefore, compositionally) the bindings obtained at different stages in the computation. In this way we obtain a full equivalence result. Another basic difference with respect to the previous approach is to annotate the variables instead of the data constructors. This allows us to extend the unification theory [10, 17] in order to deal with input-constraints in a formal way. We also give an extended algorithm for the computation of the (extended) most general unifier. Moreover, we introduce the notion of parallel composition of substitutions, which allows us to model the combination of the substitutions computed by and-parallel processes.

Other compositional models for the success set are presented in [22] and [20]. Both these approaches are based on streams of input/output *simple* substitutions, where *simple* means that the bindings are of the form $x/y$ or $x/f(x_1, \ldots, x_n)$. This restriction introduces additional complications for modeling the full unification mechanism. Thanks to our extended unification theory, we deal directly with (general) substitutions, and the correspondence with the operational semantics is therefore simpler and more intuitive.

Finally we consider the problem of also characterizing in addition to the success set, the finite failures and the infinite computations in a compositional way. It turns out that not only the streams of the operational semantics offer too little structure, but even the sequences introduced in the declarative semantics are not powerful enough. In order to model the failure set, we need not only to distinguish between external and internal computation steps, but also between different points of nondeterministic choice.

Let us illustrate how the absence of branching information causes our operational semantics to be *non*compositional. Consider the programs

$$P_1 = \{p(x) \leftarrow |q(x)., p(x) \leftarrow |r(x)., q(a) \leftarrow .r(b) \leftarrow |. s(a) \leftarrow |.\},$$

$$P_2 = \{p(x) \leftarrow |q(x)., q(a) \leftarrow |. q(b) \leftarrow |. s(a) \leftarrow |.\},$$

with mode declarations, again taken from PARLOG, $D_1 = \{p(?), q(?), r(?), s(\hat{})\}$, and $D_2 = \{p(?), q(?), s(\hat{})\}$, respectively. Consider the goal $\leftarrow p(y)$. Operationally, in both $P_1$ and $P_2$ it will suspend waiting for a binding on $y$ (either $a$ or $b$). However, if we extend the goal with an atom $s(y)$, thus yielding the goal $\leftarrow p(y), s(y)$, then we get *different* operational meanings. In $P_1$ the goal can fail (due to the choice of the wrong clause for $p(y)$), whereas in $P_2$ it cannot.

In our approach, we encode the branching information by using trees labelled with functions representing the basic unification steps. They are elements of a complete metric space, satisfying a so-called reflexive domain equation [9, 2]. We use a *denotational* style: for every operator in the language we define a semantic operator corresponding to it. The denotational model is obtained as the (unique) fixed-point of a higher-order function, a *contraction*, on the complete metric space of semantic models. The relation between this denotational semantics and the operational one is obtained via an *abstraction operator* that identifies some denotations. Although both the declarative and the denotational semantics are based on a fixed-point construction, there are some fundamental differences: in the declarative approach, the meaning of a program is constructed in a bottom-up fashion by means of an immediate consequence operator. The denotational semantics is a function from goals to tree-like structures and embodies a top-down approach.

At present, we are investigating the precise relationship between the declarative and the denotational approach. An equivalence result for the basic case of HCL can be found in [8].

## 2. The language

To describe the syntax of the language we study, we introduce the following sets:

- The set of atoms, with typical elements $A$, $B$, $H$, we denote by *Atom*.
- The set of conjunctions, with typical elements $\bar{A}$, $\bar{B}$, $\bar{G}$, we denote by *Conj*.
- The set of goals, with typical elements $\leftarrow\bar{A}$, $\leftarrow\bar{B}$, $\leftarrow\bar{G}$, we denote by *Goal*.
- The set of clauses, with typical element $C$, we denote by *Clause*.
- The set of programs, with typical element $W$, we denote by *Prog*.

Conjunctions are of the form: $\bar{A} = A_1, \ldots, A_n$. A special element in *Conj* is *true*, denoting the empty conjunct. With $\square$ we denote the goal $\leftarrow$*true*. A clause is of the form $C = H \leftarrow \bar{G} \mid \bar{B}$, where $H$, $\bar{G}$ and $\bar{B}$ are called the head, the guard, and the body of the clause, respectively. The symbol | is called the commit operator. We do not consider operators (like ;) that impose any ordering on clauses. Every program $W$ consists of a finite set of clauses together with a so-called *mode declaration*, which specifies for every predicate, which of its arguments are *input* and *output*. They are indicated by the symbols ? and ^ respectively. So, for instance, the declaration $p(?, ?, \hat{})$ specifies that the first two arguments of $p$ are *input* and the third one is *output*.

An atom $A$ in a goal is seen as an (AND-)process. Its computation proceeds by looking for a *candidate clause* in $W$. A clause is candidate if its head $H$ *input-unifies* with $A$ (i.e., the input arguments unify) and the computation of the guard succeeds, both without binding the (variables in the) input arguments of $A$. If there are candidate clauses, then the computation of $A$ *commits* to one of them (i.e., no backtracking will take place), the *output-unification* is performed and $A$ is replaced by the body of the clause. If no clauses are candidate but there are *suspended* clauses (i.e., clauses in which the input unification would succeed and bind the input-arguments), then the computation of $A$ *suspends*, and will be resumed when its (input) arguments get bound by other processes in the goal. If a guard would succeed by binding the input-arguments (of $A$), then an *error* is generated (*unsafe guard*). If none of these cases applies, then the process $A$ and the whole goal *fail*. Of course, a failure occurs also when all the processes in the goal get suspended (*deadlock*).

To simplify the discussion, we do not deal with the *error* case. More precisely, we include this case into the suspension case. So, we consider a suspension mechanism similar to the one of GHC, namely: *a clause suspends if either the input-unification or the goal evaluation would instantiate the input-arguments of A.*

## 3. Operational semantics

For the rest of the paper let $W$ denote a fixed program. The set of variables occurring in a conjunction $\bar{A}$ is indicated by $\mathcal{V}(\bar{A})$. We postulate a function *invar* that gives for every atom $A$ the set of variables occurring in those arguments of $A$ that are specified as input by the mode declaration of $W$. Given a set of variables $V$, $W_V$ denotes the program whose clauses are *variants* (see [17]), in which there occur only variables *not* belonging to $V$, of the clauses of $W$. We introduce the set of substitutions *Subst*, with typical elements $\vartheta$, $\gamma \in Subst$. $\varepsilon$ is the *empty substitution.* For $V$ a finite set of variables, we use $\vartheta_{|V}$ to denote the restriction of $\vartheta$ to $V$. Further we have the familiar notion of *mgu*, which is a partial function from pairs of atoms to substitutions. We introduce the notions of *input* and *output* mgus. Consider two atoms $A = p(t_1, \ldots, t_n)$ and $A' = p(t'_1, \ldots, t'_n)$. Assume that the declaration-mode of $p$ has the symbol ? (input-mode) on the arguments $i_1, \ldots i_k$. Then, $mgu_i(A, A')$ denotes $mgu(\{\{t_{i_1}, t'_{i_1}\}, \ldots \{t_{i_k}, t'_{i_k}\}\})$. In a similar way we define $mgu_o(A, A')$ to be the *mgu* of the output arguments.

The operational semantics will be based on the following *transition relation*:

**Definition 3.1** (*Transition relation*). Let $\rightarrow \subseteq (Goal \times Subst) \times (Goal \times Subst)$ be the smallest relation satisfying

(1) If $\exists H \leftarrow \bar{G} \mid \bar{B} \in W_{\nu(A)}, \exists mgu_i(A\vartheta, H)$
$[\langle \leftarrow \bar{G}, mgu_i(A\vartheta, H)\rangle \xrightarrow{*} \langle \Box, \vartheta' \rangle$, and $\vartheta'|_{invar(A\vartheta)} = \varepsilon]$,
then $\langle \leftarrow A, \vartheta \rangle \rightarrow \langle \leftarrow outunif(A\vartheta, H\vartheta'), \bar{B}, \vartheta\vartheta' \rangle$.

(2) If $\exists mgu_o(A\vartheta, H\vartheta')$,
then $\langle \leftarrow outunif(A\vartheta, H\vartheta'), \bar{B}, \vartheta' \rangle \rightarrow \langle \leftarrow \bar{B}, \vartheta' mgu_o(A\vartheta, H\vartheta') \rangle$.

(3) If $\langle \leftarrow \bar{A}, \vartheta \rangle \rightarrow \langle \leftarrow \bar{A}', \vartheta' \rangle \mid \langle \Box, \vartheta' \rangle$
then $\langle \leftarrow \bar{A}, \bar{B}, \vartheta \rangle \rightarrow \langle \leftarrow \bar{A}', \bar{B}, \vartheta' \rangle \mid \langle \leftarrow \bar{B}, \vartheta' \rangle$
$\langle \leftarrow \bar{B}, \bar{A}, \vartheta \rangle \rightarrow \langle \leftarrow \bar{B}, \bar{A}', \vartheta' \rangle \mid \langle \leftarrow \bar{B}, \vartheta' \rangle$.

In these transitions, $\vartheta$ represents the substitution that has been computed until that moment. In (1), it is stated that we can resolve $\leftarrow A$ if we can find a (renamed) clause in our program with a head $H$ that can be input-unified with $A$; moreover, the refutation of the guard $\bar{G}$ of that clause must terminate successfully and the total substitution $\vartheta'$ must not instantiate any input variables of $A\vartheta$. The output-unification *outunif*$(A\vartheta, H\vartheta')$ should be possibly computed in parallel with the other atoms in the goal. A conjunction, in (3), is evaluated by the parallel execution of its conjuncts, modelled here by interleaving. In the following definition we give the operational semantics.

**Definition 3.2** (*Operational semantics*). We define

$$\mathcal{O}_1: \mathit{Goal} \to M_1, \quad \text{with } M_1 = \mathcal{P}(\mathit{Subst})$$

$$\mathcal{O}_2: \mathit{Goal} \to M_2, \quad \text{with } M_2 = \mathcal{P}(\mathit{Subst}_\delta^\infty).$$

(Here $\mathit{Subst}_\delta^\infty = \mathit{Subst}^+ \cup \mathit{Subst}^\omega \cup \mathit{Subst}^*.\{\delta\}$, with typical element $\vartheta_1. \cdots \vartheta_n. \cdots$; the symbol $\delta$ denotes failure; $\mathcal{P}(X)$ is the set of all the subsets of $X$.)

We put $\mathcal{O}_i[\![\leftarrow \mathit{true}]\!] = \{\varepsilon\}$, and

$$\mathcal{O}_1[\![\leftarrow \bar{A}]\!] = \{\vartheta_{|\mathcal{V}(\bar{A})} \mid \langle \leftarrow \bar{A}, \varepsilon\rangle \xrightarrow{*} \langle \Box, \vartheta\rangle\};$$

$$\begin{aligned}
\mathcal{O}_2[\![\leftarrow \bar{A}]\!] = {} & \{(\vartheta_1. \cdots \vartheta_n)_{|\mathcal{V}(\bar{A})} \in \mathit{Subst}^+ \mid \\
& \qquad \langle \leftarrow \bar{A}, \varepsilon\rangle \to \langle \leftarrow \bar{A}_1, \vartheta_1\rangle \to \cdots \to \langle \Box, \vartheta_n\rangle\} \\
& \cup \{(\vartheta_1. \cdots \vartheta_n)_{|\mathcal{V}(\bar{A})}.\delta \in \mathit{Subst}^*.\{\delta\} \mid \\
& \qquad \langle \leftarrow \bar{A}, \varepsilon\rangle \to \cdots \to \langle \bar{A}_n, \vartheta_n\rangle \not\to \wedge \leftarrow \bar{A}_n \neq \Box\} \\
& \cup \{(\vartheta_1. \cdots)_{|\mathcal{V}(\bar{A})} \in \mathit{Subst}^\omega \mid \langle \leftarrow \bar{A}, \varepsilon\rangle \to \langle \leftarrow \bar{A}_1, \vartheta_1\rangle \to \cdots\}.
\end{aligned}$$

The *success set* for $\leftarrow \bar{A}$ is given by $\mathcal{O}_1[\![\leftarrow \bar{A}]\!]$: it contains all computed answer substitutions corresponding to all successfully terminating computations. In addition, the set $\mathcal{O}_2[\![\leftarrow \bar{A}]\!]$ takes into account all failing and infinite computations, represented by elements of $\mathit{Subst}^*.\{\delta\}$ and Subst$^\omega$, respectively. The relation between $\mathcal{O}_1$ and $\mathcal{O}_2$ is obvious. If we set

$$\mathit{last}(X) = \{\vartheta \mid \exists s \in \mathit{Subst}^* \, (s.\vartheta \in X)\},$$

then we have: $\mathcal{O}_1 = \mathit{last} \circ \mathcal{O}_2$.

In the following sections, $\mathcal{O}_1$ and $\mathcal{O}_2$ will be related to a declarative and a denotational semantics, respectively.

We did not include all deadlocking and infinite behaviours in $\mathcal{O}_2$. In fact, we omitted the so called local deadlock in guards. This can appear when a local computation commits to "wrong" clauses. It is not too difficult to adapt $\mathcal{O}_2$, but we prefer not to do so because it obscures the equivalence proof between the denotational model and $\mathcal{O}_2$. Moreover, on FCP the models coincide.

## 4. Declarative semantics

We define the declarative (fixpoint) semantics of the language described in the previous section. We make use of an extended notion of Herbrand base and interpretations, enriched with variables (that are used for modelling the notion of *computed substitution* [19, 11, 12]) and *annotations* (that are used for modelling the synchronization mechanism of concurrent logic languages, see [18] and [19] for similar approaches). We extend the standard notions of the unification theory [10, 17] in a formal framework. Moreover, we introduce the notion of *parallel composition*, that allows us to formalize the combination (plus consistency check) of the substitutions computed by subgoals run in parallel. Finally, we introduce the notion of *sequences of substitutions*, that allow us to overcome the difficulties presented in [19] concerning deadlock. (The construction is essentially the one that was used for a declarative semantics of Guarded Horn Clauses, given in [7, 21].)

### *4.1. Annotated variables*

In order to model the synchronization mechanism of the language we introduce the notion of *annotated variable*. The annotation can occur on a variable in the goal, and it means that such a variable is in an input-argument and therefore cannot be bound, during the derivation step, before commitment. In other words, such a variable can receive bindings from the execution of other atoms in the goals, but cannot produce bindings by the execution of the atom in which it occurs (before commitment).

We will denote the set of variables, with typical elements $x, y, \ldots$, by *Var*, and the set of the annotated variables, with typical elements $x^-, y^-, \ldots$, by $Var^-$. From a mathematical point of view, we can consider "$^-$" as a bijective mapping $^-: Var \to Var^-$. The elements of $Var \cup Var^-$ will be represented by $v, w, \ldots$ . The set of terms *Term*, with typical element $t$, is extended on $Var \cup Var^-$. The term $t^-$ is obtained by replacing in $t$ every variable $x \in Var$ by $x^-$. The set of variables occurring in the term $t$ is denoted by $\mathcal{V}(t)$.

The notion of substitution extends naturally to the new set of variables and terms. Namely, a substitution $\vartheta$ is a mapping $\vartheta: Var \cup Var^- \to Term$, such that $\vartheta(v) \neq v$ for finitely many $v$ only. $\vartheta$ will be represented by the set $\{v/t \mid v \in Var \cup Var^- \wedge \vartheta(v) = t \neq v\}$. In order to model the difference between producing and receiving a binding we introduce an asymmetry in the definition of the application of a substitution $\vartheta$ to a term (or atom, or formula) $t$:

$$t\vartheta = \begin{cases} \vartheta(x) & \text{if } t = x \in Var, \\ \vartheta(x^-) & \text{if } x^- \in Var^- \text{ and } \vartheta(x^-) \neq x^-, \\ \vartheta(x)^- & \text{if } t = x^- \in Var^- \text{ and } \vartheta(x^-) = x^-, \\ f(t_1\vartheta, \ldots, t_n\vartheta) & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

The new notion of application differs from the standard one in that $\{v \in Var \cup Var^- \mid \vartheta(v) \neq v\}$ (the set of variables mapped by $\vartheta$ to a different term) is now a subset of $\{v \in Var \cup Var^- \mid v\vartheta \neq v\}$ (the set of variables bound by $\vartheta$ to a different term). An annotated variable bound to a different term represents the ability to receive a binding from the computation of another atom in the goal.

We factorize the set of substitutions with respect to the equivalence relation $\vartheta_1 \equiv \vartheta_2$ iff $\forall v \in Var \cup Var^-$ $[v\vartheta_1 = v\vartheta_2]$. From now on, a substitution $\vartheta$ will indicate its equivalence class.

**Example 4.1.** Consider the atom $A = p(f(x, y), x, y)$. We annotate the variables in $A$ to obtain $A^- = p(f(x^-, y^-), x^-, y^-)$. Consider now the substitution $\vartheta = \{x/g(z), y/h(w), y^-/h(a)\}$. We have: $A^-\vartheta = p(f(g(z^-), h(a)), g(z^-), h(a)))$.

The notion of composition $\vartheta_1\vartheta_2$, of two substitutions, $\vartheta_1$ and $\vartheta_2$ is extended as follows

$$\forall v \in Var \cup Var^- \quad [v(\vartheta_1\vartheta_2) = (v\vartheta_1)\vartheta_2].$$

The composition is associative and the empty substitution $\varepsilon$ is the neutral element.

We extend the notions of domain and co-domain of a substitution in order to deal with the new notion of application:

$$\mathscr{D}(\vartheta) = \{x \in Var \cup Var^- \mid x\vartheta \neq x\},$$

$$\mathscr{C}(\vartheta) = \bigcup_{x \in \mathscr{D}(\vartheta)} \mathscr{V}(x\vartheta).$$

$\vartheta$ is called *idempotent* iff $\vartheta\vartheta = \vartheta$, or, equivalently, iff $\mathscr{C}(\vartheta) \cap \mathscr{D}(\vartheta) = \emptyset$.

Given a set of sets of terms $M$, we define $\vartheta$ to be a unifier for $M$ iff

$$\forall S \in M\, \forall t_1, t_2 \in S \quad [t_1\vartheta = t_2\vartheta \text{ and } t_1^-\vartheta = t_2^-\vartheta].$$

The ordering on substitutions is the standard one, namely: $\vartheta_1 \leq \vartheta_2$ iff $\exists \vartheta_3$ $[\vartheta_1\vartheta_3 = \vartheta_2]$ ($\vartheta_1$ *is more general than* $\vartheta_2$). The set of idempotent mgus (most general unifiers) of a set of sets of terms $M$ is denoted by $mgu(M)$.

In the appendix we give an extended version of the unification algorithm based on the one presented in [1].

## *4.2. Parallel composition of substitutions*

In this section we introduce the notion of *parallel composition* of substitutions and of sets of substitutions, both denoted by $\hat{\circ}$. Intuitively, the parallel composition is meant to be the formalization of one of the basic operations performed by the *parallel execution model* of logic programs. When two atoms $A_1$ and $A_2$ (in the same goal) are run in parallel, the associated computed answer substitutions $\vartheta_1$ and $\vartheta_2$ have to be combined afterwards in order to get the final result. This operation can be performed in the following way: consider the set of all the pairs corresponding to the bindings of both $\vartheta_1$ and $\vartheta_2$. Then compute the most general unifier of such a set. Note that the *consistency check* corresponds to a verification that such a set is unifiable.

**Definition 4.2.** In the following, $\mathscr{S}(\vartheta)$ is the set of sets $\{\{x, t\} \mid x/t \in \vartheta\}$. $\Theta_1, \Theta_2$ are sets of substitutions.

(1) $\vartheta_1 \mathbin{\hat{\circ}} \vartheta_2 = mgu(\mathscr{S}(\vartheta_1) \cup \mathscr{S}(\vartheta_2))$.

(2) $\Theta_1 \mathbin{\hat{\circ}} \Theta_2 = \bigcup_{\vartheta_1 \in \Theta_1, \vartheta_2 \in \Theta_2} \vartheta_1 \mathbin{\hat{\circ}} \vartheta_2$.

We will denote the sets $\{\vartheta\} \mathbin{\hat{\circ}} \Theta$ and $\Theta \mathbin{\hat{\circ}} \{\vartheta\}$ by $\vartheta \mathbin{\hat{\circ}} \Theta$ and $\Theta \mathbin{\hat{\circ}} \vartheta$, respectively.

**Example 4.3.** (1) Consider the program $\{p(f(a)) \leftarrow |., q(f(a)) \leftarrow |.\}$, with declaration-mode $\{p(?), q(\hat{\ })\}$, and consider the goal $\leftarrow p(x), q(x)$. We annotate the variable $x$, in $p(x)$, in order to express the input-mode constraint. We have

$$mgu(p(x^-), p(f(a))) = \{\vartheta_1\}, \quad \text{where } \vartheta_1 = \{x^-/f(a)\}, \quad \text{and}$$

$$mgu(q(x), q(f(a))) = \{\vartheta_2\}, \quad \text{where } \vartheta_2 = \{x/f(a)\}.$$

Now observe that since $\vartheta_1 \in mgu(\mathscr{S}(\vartheta_1))$, $\vartheta_2 \in mgu(\mathscr{S}(\vartheta_2))$ and $\vartheta_1 \leq \vartheta_2$ we have $\vartheta_2 \in \vartheta_1 \mathbin{\hat{\circ}} \vartheta_2$.

(2) Consider now the same program and goal as before, but let the declaration mode by $\{p(?), q(?)\}$. We have

$$mgu(p(x^-), p(f(a))) = mgu(q(x^-), q(f(a))) = \{\vartheta_1\},$$

and $\vartheta_1 \in \vartheta_1 \mathbin{\hat{\circ}} \vartheta_1$, whilst $\vartheta_2 \notin \vartheta_1 \mathbin{\hat{\circ}} \vartheta_1$.

In (1) the goal can be refuted by a suitable ordering on the execution of the atoms ($q(x)$ before $p(x)$). This corresponds to getting a substitution $\vartheta_2$, that does not bind any annotated (i.e., input-constrained) variable. This is not the case in (2), and indeed no refutations are possible.

## *4.3. Sequences of substitutions*

As shown in [11, 12], the computed bindings in HCL can be declaratively modelled by using a not ground Herbrand Base, or equivalently, a set of pairs consisting of an atom and a substitution. However, when the input-constraints are present, it is not sufficient to consider only a substitution. In fact, as shown in [18] and [19], a *flat* representation of the computed bindings is not powerful enough to model compositionally the *results* of the possible interleavings in the executions of the atoms in a goal. We have to *register* the whole history of the execution of the atom, and therefore we have to deal with *sequences of substitutions.* Since we model only the success set, we need to consider only finite sequences. However, the set $Subst^+$ used for the operational semantics is still too weak a structure. To represent the *critical sections* given by the input-unification and the guard evaluation, we need to separate a subsequence from the rest.

**Definition 4.4.** The finite sequences of substitutions, with typical element $s$, are defined by the following (abstract) syntax

$$s \ ::= \ \vartheta \mid [s]_V \mid s_1.s_2.$$

The role of the square brackets is to mark the beginning and the end of a *critical section.* $V$ represents a set of variables, whose annotation has to be removed when computing the result of a sequence of substitutions. Their meaning will be clarified by the definition of the *interleaving operator* and *result operator.* We introduce the following notations. If $S$ and $S'$ are sets of sequences, then $S.S' \overset{\text{def}}{=} \{s.s' \mid s \in S, s' \in S'\}$ and $[S]_V \overset{\text{def}}{=} \{[s]_V \mid s \in S\}$. If $s = \vartheta'.s'$, then $\vartheta \mathbin{\hat{\circ}} s \overset{\text{def}}{=} (\vartheta \mathbin{\hat{\circ}} \vartheta').s'$ and $\vartheta \mathbin{\hat{\circ}} ([s]_V.s'') \overset{\text{def}}{=} [\vartheta \mathbin{\hat{\circ}} \vartheta').s']_V.s''$. For $\Theta$ a set of substitutions we have $\Theta \mathbin{\hat{\circ}} s \overset{\text{def}}{=} \bigcup_{\vartheta \in \Theta} \vartheta \mathbin{\hat{\circ}} s$. The length $\#(s)$ of $s$ is defined as follows:

- $\#(\vartheta) = 1$,
- $\#([s]) = \#(s)$,
- $\#(s_1.s_2) = \#(s_1) + \#(s_2)$.

If all the elements of $S$ have the same length, i.e. $\exists k \colon \forall s \in S \colon \#(s) = k$, then we define $\#(S) = k$.

**Definition 4.5** (*Interleaving operator*)

$$(1) \quad s_1 \parallel s_2 = (s_1 \mathbin{⫴} s_2) \cup (s_2 \mathbin{⫴} s_1),$$

$$(\vartheta.s_1) \mathbin{⫴} s_2 = \vartheta.(s_1 \parallel s_2),$$

$$([s]_V.s_1) \mathbin{⫴} s_2 = [s]_V.(s_1 \parallel s_2).$$

$$(2) \quad S_1 \parallel S_2 = \bigcup_{s_1 \in S_1, s_2 \in S_2} s_1 \parallel s_2.$$

Since the interleaving operator is associative we can omit parentheses.

The following definition introduces the notion of *result* $\mathcal{R}$ of a sequence $s$ (or a set of sequences $S$) of substitutions. Roughly, such a result is obtained by performing the parallel composition of each element of the sequence with the next one, and by checking, each time, that the partial result does not violate input-mode constraints.

**Definition 4.6**

$$(1) \quad \mathcal{R}(\vartheta) = \begin{cases} \{\vartheta\} & \text{if } \vartheta_{|Var^-} = \varepsilon, \\ \emptyset & \text{otherwise.} \end{cases}$$

(2) $\mathcal{R}([s]_V) = disann_V(\mathcal{R}(s))$ where $disann_V(s)$ removes all the annotations of the variables of $V$ which occur in $s$.

(3) $\mathcal{R}(s_1.s_2) = \mathcal{R}(\mathcal{R}(s_1) \mathbin{\hat{\circ}} s_2)$.

(4) For $S$ a set of sequences we define $\mathcal{R}(S) = \bigcup_{s \in S} \mathcal{R}(s)$.

Thus, rule (2) specifies that, after a critical section, the input-constraints are released. Rule (1) checks that $\vartheta$ (to be intended as the partial result) does not map annotated variables. Rule (3) specifies the order of evaluation of a sequence: from left to right. Indeed, we have $\mathcal{R}(\vartheta_1.\vartheta_2.\cdots.\vartheta_n) = \mathcal{R}(\ldots\mathcal{R}(\mathcal{R}(\vartheta_1) \mathbin{\hat{\circ}} \vartheta_2)\ldots \mathbin{\hat{\circ}} \vartheta_n)$.

## 4.4. *Least fixpoint semantics*

In this section we introduce the notion of interpretation, and we define a continuous mapping (associated to the program) on interpretations. The least fixpoint of this mapping will be used to define the fixpoint semantics. Such a mapping is the extension of the *immediate consequence* operator for HCL [30, 1].

The *Herbrand base with variables* associated to the program $W$, denoted by $\mathcal{B}$, consists of all the possible atoms that can be obtained by applying the predicates of $W$ to the elements of *Term*. An interpretation $I$ of $W$ is a set of pairs of the form $\langle A, s\rangle$, where $A$ is an atom in $\mathcal{B}$ and $s$ is a sequence of substitutions on *Var* and *Term*. $\langle A, s\rangle \in I$ can be read declaratively as *$A$ is true in $I$ under the sequence $s$.* We remark the similarity with temporal logic, although we do not investigate this relation here. $\mathcal{I}$ will denote the set of all the interpretations of $W$.

$\mathcal{I}$ is a complete lattice with respect to the set-inclusion, where the empty set $\emptyset$ is the minimum element, and the *set union* $\cup$ and the *set intersection* $\cap$ are the *sup* and *inf* operations, respectively.

The following definition, which will be used in the least fixpoint construction, is introduced mainly for technical reasons.

**Definition 4.7.** Let $s_1, \ldots, s_h$ be sequences of substitutions, and let $A_1, \ldots, A_k$ $(h \leq k)$ be atoms. $s_1, \ldots, s_h$ are *locally independent* on $A_1, \ldots A_k$ in

$$\forall s_i, \forall \vartheta \text{ in } s_i\colon \quad (\mathcal{D}(\vartheta) \cup \mathcal{C}(\vartheta)) \cap \mathcal{V}(A_1, \ldots, A_k) \subseteq \mathcal{V}(A_i).$$

In the following, we use the notation $\bar{s}$ to denote a sequence of sequences of substitutions $s_1, \ldots, s_n$. If moreover $\bar{A} = A_1, \ldots, A_n$, then $\langle \bar{A}, \bar{s}\rangle$ stands for $\langle A_1, s_1\rangle, \ldots, \langle A_n, s_n\rangle$, and $\|\ (\bar{s})$ stands for $s_1 \| \cdots \| s_n$.

**Definition 4.8.** The mapping $T\colon \mathcal{I} \to \mathcal{I}$, associated to $W$, is defined as follows:

$$\begin{aligned} T(I) = \{\langle A, s\rangle \mid \exists H \leftarrow \bar{G} \mid \bar{B} \in W_{\mathcal{V}(A)},\\ &\exists \bar{s}', \bar{s}'' \text{ locally independent on } \bar{G}, \bar{B}, A,\\ &\langle \bar{G}, \bar{s}'\rangle, \langle \bar{B}, \bar{s}''\rangle \in I\colon\\ &s \in [mgu_i(A^-, H).(\|(\bar{s}'))]_V.mgu_0(A, H).(\|(\bar{s}''))\}. \end{aligned}$$

In this definition $V$ stands for $\mathcal{V}(A^-, H, \bar{s}')$. A possible sequence for $A$ results from the critical section containing the $mgu_i$ with the head of a clause, and a sequence resulting from the guard. The input variables in $A$ are annotated. The whole is followed by the $mgu_0$ and a sequence resulting from the body.

The following proposition is an immediate extension of the corresponding classical result.

**Proposition 4.9.** *$T$ is continuous. Thus its least fixpoint $lfp(T)$ exists, and $lfp(T) = \bigcup_{n\geq 0} T^n(\emptyset)$ holds.*

We define the least fixpoint semantics associated to $W$ as the set $\mathscr{F}(W) = lfp(T)$.

### *4.5. Equivalence results*

In this subsection we prove the equivalence between the declarative semantics and the operational semantics. The equivalence is restricted to the *success case*, namely, to the substitutions computed by a refutation. We will show that

$$\begin{aligned}&\mathscr{O}_1(\leftarrow\bar{A})\\ &\quad = \{\vartheta \mid \exists \bar{s} \text{ locally independent on } \bar{A}\colon \langle \bar{A}, \bar{s}\rangle \in \mathscr{F}(W) \text{ and } \vartheta \in \mathscr{R}(\|(\bar{s})_{|\mathscr{V}(\bar{A})}\}.\end{aligned}$$

The following proposition can easily be proved for the unification algorithm given in Definition 8.1.

**Proposition 4.10.** *Let $M_1, M_2, M_3$ be sets of terms. Let $\vartheta_1 \in mgu(M_1)$ and $\vartheta_2 \in mgu(M_2)$. Then*

(1) $mgu(M_1 \cup M_2) = \vartheta_1\ mgu(M_2\vartheta_1) = \vartheta_2\ mgu(M_1\vartheta_2)$,

(2) $mgu(M_1 \cup M_2^- \cup M_3) = \vartheta_1\ mgu(M_2\vartheta_1^- \cup M_3\vartheta_1)$.

**Lemma 4.11.** *Let $A, H$ be atoms. Let $\mu$ be an idempotent positive substitution, i.e., binding only non-annotated variables, with no variables in common with $H$. Then*

$$\mu\ mgu_i((A\mu)^-, H) = \mu \mathbin{\hat{\circ}} mgu_i(A^-, H).$$

**Proof.** It is a particular case of Proposition 4.10(2). □

**Lemma 4.12.** *Let $\mu$ be a positive substitution. Let $\vartheta_1, \ldots, \vartheta_n$ be idempotent substitutions. We have:*

$$(\mathscr{D}(\mu) \cap (\mathscr{D}(\vartheta_i) \cup \mathscr{C}(\vartheta_i)) = \emptyset, \quad i = 1, \ldots, n$$

*then*

$$\mu\mathscr{R}(\vartheta_1 . \cdots . \vartheta_n) = \mathscr{R}((\mu\vartheta_1). \cdots .(\mu\vartheta_n)).$$

**Proof.** Immediate. □

**Lemma 4.13.** *Let $W$ be a program. Let $\bar{A}$ be a sequence of goals, and let $\rho$ be a positive renaming such that $\bar{A}\rho$ is a variant of $\bar{A}$. Then*

$$\exists \bar{s} \text{ locally independent on } \bar{A}\text{: } \langle \bar{A}, \bar{s}\rangle \in T^k_W(\emptyset)$$

$$\Leftrightarrow$$

$$\exists \bar{s}' \text{ locally independent on } \bar{A}\rho\text{: } \langle \bar{A}\rho, \bar{s}'\rangle \in T^k_W(\emptyset) \text{ and}$$
$$\mathcal{R}(\rho.Int(\bar{s}))_{|\mathcal{V}(\bar{A})\cup\mathcal{D}(\rho)} = \rho\mathcal{R}(Int(\bar{s}'))_{|\mathcal{V}(\bar{A})\cup\mathcal{D}(\rho)} \text{ and}$$
$$\#(Int(\bar{s})) = \#(Int(\bar{s}')).$$

**Proof.** Let $\bar{A} = A_1, \ldots, A_n$, and $\bar{s} = s_1, \ldots, s_n$. For each $i = 1, \ldots, n$, let $s_i = \vartheta_i.s_i''$ and let $H_i$ be the head of the clause used to obtain $\langle A_i, s_i\rangle \in T^k_W(\emptyset)$. Then, define $s_i' = \vartheta_i'.s_i'''$, where $\vartheta_i' \in mgu_i(A_i\rho^-, H_i)$, and $s_i'''$ is the renamed version of $s_i''$ (in order to meet the requirement of local independence). Then we have $\langle A_1\rho, s_1'\rangle, \ldots, \langle A_n\rho, s_n'\rangle \in T^k_W(\emptyset)$. Moreover by Lemmas 4.11 and 4.12 we have, for each $s \in (s_1 \| \cdots \| s_n)$,

$$\mathcal{R}(\rho.s)_{|\mathcal{V}(\bar{A})\cup\mathcal{D}(\rho)} = (\rho\mathcal{R}(s'))_{|\mathcal{V}(\bar{A})\cup\mathcal{D}(\rho)} \tag{1}$$

for an appropriate $s' \in (s_1' \| \cdots \| s_n')$. Analogously, for each $s' \in (s_1' \| \cdots \| s_n')$, there exists an appropriate $s \in (s_1 \| \cdots \| s_n)$ such that equality (1) holds. □

**Lemma 4.14.** *Let $W$ be a program. Let $\bar{A}$ be a sequence of atoms, and let $\mu$ be a positive substitution. Then*

$$\exists \bar{s} \text{ locally independent on } \bar{A}\text{: } \langle \bar{A}, \bar{s}\rangle \in T^k_W(\emptyset)$$

$$\Leftrightarrow$$

$$\exists \bar{s}' \text{ locally independent on } \bar{A}\mu\text{: } \langle \bar{A}\mu, \bar{s}'\rangle \in T^k_W(\emptyset) \text{ and}$$
$$\mathcal{R}(\mu.\mathrm{Int}(\bar{s}))_{|\mathcal{V}(\bar{A})\cup\mathcal{D}(\mu)} = \mu\mathcal{R}(Int(\bar{s}'))_{|\mathcal{V}(\bar{A})\cup\mathcal{D}(\mu)} \text{ and}$$
$$\#(Int(\bar{s})) = \#(Int(\bar{s}')).$$

**Proof.** ($\Rightarrow$): Let $\mu$ be a positive substitution, and let $\rho$ be a renaming on $\mathcal{D}(\mu)$ such that $(\bar{A})\rho\mu\rho^{-1} = \bar{A}$. Define $\bar{s}' = s_1', \ldots, s_n'$ as in Lemma 4.13, apart from the first element of each $s_i'$ that is chosen in $mgu_i(A_i\mu^-, H_i)$. Then we have

(1) *$\bar{s}'$ is locally independent on $\bar{A}\mu$.*
(2) $\langle \bar{A}\mu, \bar{s}'\rangle \in T^k_W(\emptyset)$.
(3) $(\mu\mathcal{R}(Int(\bar{s}')))_{|\mathcal{V}(A)\cup\mathcal{D}(\mu)}$
(by Lemmas 4.11 and 4.12)
$= (\mathcal{R}(Int(\mu\bar{s}')))_{|\mathcal{V}(A)\cup\mathcal{D}(\mu)}$
(since only the $\vartheta_i'$'s $\in mgu_i(A_i\mu^-, H_i)$
have variables in common with $\mu$)

$= (\mathcal{R}(\cdots \| (\mu\, mgu_i(A_i\mu^-, H_i)) \|$

$\quad mgu_0(A_i\mu^-, H_i) \| \cdots \| Int(\bar{s}''')))_{|\mathcal{V}(A)\cup\mathcal{D}(\mu)}$

$= (\mathcal{R}(\cdots \| (\mu\, mgu_i(A_i\mu^-, H_i)) \|$

$\quad mgu_0(A_i\mu^-, H_i) \| \cdots \| Int(\bar{s}''')))_{|\mathcal{V}(A)\cup\mathcal{D}(\mu)}$

(since the $s_i'''$'s have no variables in common with $\rho$)

$= (\mathcal{R}(\cdots \| ((\rho\rho^{-1}\mu) \hat{\circ}\, mgu_i(A_i\rho^-, H_i)) \|$

$\quad mgu_0(A_i\rho^-, H_i) \| \cdots \| Int(\rho\bar{s}''')))_{|\mathcal{V}(A)\cup\mathcal{D}(\mu)}$

(by Lemmas 4.11 and 4.12)

$= (\rho(\mathcal{R}(\cdots \| ((\rho^{-1}\mu) \hat{\circ}\, mgu_i(A_i\rho^-, H_i)) \|$

$\quad mgu_0(A_i\rho^-, H_i) \| \cdots \| Int(\bar{s}''')))))_{|\mathcal{V}(A)\cup\mathcal{D}(\mu)}$

(by lemma 4.13)

$= (\mathcal{R}(\cdots \| (\rho.(\rho^{-1}\mu).mgu_i(A_i\rho^-, H_i)) \|$

$\quad mgu_0(A_i\rho^-, H_i) \| \cdots \| Int(\bar{s}'')))))_{|\mathcal{V}(A)\cup\mathcal{D}(\mu)}$

(since $\rho_{-1}\mu = \rho_{-1} \hat{\circ}\, \mu$)

$= (\mathcal{R}(\cdots \| (\rho.\rho^{-1}.\mu.mgu_i(A_i\rho^-, H_i)) \|$

$\quad mgu_0(A_i\rho^-, H_i) \| \cdots \| Int(\bar{s}'')))))_{|\mathcal{V}(A)\cup\mathcal{D}(\mu)}$

$= (\mathcal{R}(\cdots \| (\mu.mgu_i(A_i\rho^-, H_i)) \| mgu_0(A_i\rho^-, H_i) \|$

$\quad \cdots \| Int(\bar{s}'')))))_{|\mathcal{V}(A)\cup\mathcal{D}(\mu)}$

$= (\mathcal{R}(\mu.(Int(\bar{s}'))))_{|\mathcal{V}(A)\cup\mathcal{D}(\mu)}.$

($\Leftarrow$): Analogously. $\square$

We now prove the equivalence of the fixpoint semantics we have defined and the operational semantics. We introduce the following notation.

**Definition 4.15** (*Transition relation, k steps*)

(1) If $\exists H \leftarrow \bar{G} | \bar{B} \in W_{\vartheta(A)}, \exists mgu_i(A\vartheta, H)$
$[\langle \leftarrow \bar{G}, mgu_i(A\vartheta, H)\rangle \rightarrow^k \langle \Box, \vartheta' \rangle,\ and\ \vartheta'|_{invar(A\vartheta)} = \varepsilon]$,
then$\langle \leftarrow A, \vartheta \rangle \rightarrow^{k+1} \langle \leftarrow outunif(A\vartheta, H\vartheta'), \bar{B}, \vartheta\vartheta' \rangle$.

(2) If $\exists mgu_0(A\vartheta, H\vartheta')$,
then$\langle \leftarrow outunif(A\vartheta, H\vartheta'), \bar{B}, \vartheta' \rangle \rightarrow_1 \langle \leftarrow \bar{B}, \vartheta'\, mgu_0(A\vartheta, H\vartheta') \rangle$.

(3) If $\langle \leftarrow \bar{A}, \vartheta \rangle \rightarrow^k \langle \leftarrow \bar{A}', \vartheta' \rangle | \langle \Box, \vartheta' \rangle$
then$\langle \leftarrow \bar{A}, \bar{B}, \vartheta \rangle \rightarrow^k \langle \leftarrow \bar{A}', \bar{B}, \vartheta' \rangle | \langle \leftarrow \bar{B}, \vartheta' \rangle$
and$\langle \leftarrow \bar{B}, \bar{A}, \vartheta \rangle \rightarrow^k \langle \leftarrow \bar{B}, \bar{A}', \vartheta' \rangle | \langle \leftarrow \bar{B}, \vartheta' \rangle$.

(4) If $\langle \leftarrow \bar{A}, \vartheta \rangle \leftarrow^{k_1} \langle \leftarrow \bar{A}', \vartheta' \rangle$
and$\langle \leftarrow \bar{A}', \vartheta' \rangle \rightarrow^{k_2} \langle \leftarrow \bar{A}'', \vartheta'' \rangle$.
then$\langle \leftarrow \bar{A}, \vartheta \rangle \rightarrow^{k_1+k_2} \langle \leftarrow \bar{A}'', \vartheta'' \rangle$.

The relation $\rightarrow^k$ represents $k$ applications of the resolution step. It is easy to see that

$$\langle \leftarrow \bar{A}, \vartheta \rangle \rightarrow^+ \langle \leftarrow \bar{A}', \vartheta' \rangle$$

if and only if

$$\exists k[\langle \leftarrow \bar{A}, \vartheta \rangle \rightarrow^k \langle \leftarrow \bar{A}', \vartheta' \rangle]$$

where $\rightarrow^+$ is the transitive closure of the relation $\rightarrow$ introduced earlier.

**Theorem 4.16** (Soundness). *Let $W$ be a program, and let $\bar{A}$ be a sequence of atoms. Then*

$$\mathcal{O}_1(\leftarrow \bar{A}) \subseteq \{\vartheta \mid \exists \bar{s} \text{ locally independent on } \bar{A}\text{:}$$
$$\langle \bar{A}, \bar{s} \rangle \in \mathcal{F}(W) \text{ and } \vartheta \in \mathcal{R}(\|(\bar{s}))_{|\mathcal{V}(\bar{A})}\}.$$

**Proof.** By induction on the number of steps $k$.

($k=1$): Assume $\langle \leftarrow \bar{A}, \varepsilon \rangle \rightarrow^1 \langle \Box, \vartheta \rangle$. Then $\bar{A}$ is composed by only one atom, say $A$. In this case there exists in $W_{\mathcal{V}(A)}$ a clause of the form $H \leftarrow |.$ such that $\vartheta \in mgu_i(A, H)$ and $\vartheta_{|\mathcal{V}(A)} = \varepsilon$. Then $\langle A, \sigma \rangle \in T^1_W(\emptyset)$ holds, for each $\sigma \in mgu_i(A^-, H)$. Since $\vartheta_{|\mathcal{V}(A)} = \varepsilon$, there exists $\sigma \in mgu_i(A^-, H)$ such that $\sigma$ does not map variables in $A$, i.e., $\sigma$ is positive. Therefore $\mathcal{R}(\sigma)_{|\mathcal{V}(A)} = \{\varepsilon\}$.

($k>1$): Assume $\langle \leftarrow \bar{A}, \varepsilon \rangle \rightarrow^{k_1} \langle \bar{A}', \sigma\mu \rangle \rightarrow^{k_2} \langle \Box, \sigma\mu\vartheta' \rangle$ and $\vartheta = (\sigma\mu\vartheta')_{|\mathcal{V}(\bar{A})}$. Let $A_i$ be the atom in $\bar{A}$ selected for the first derivation step. Then, there exists a clause $H \leftarrow \bar{G} \mid \bar{B}$ in $W_{\mathcal{V}(\bar{A})}$ such that:

- $\sigma \in mgu_i(A_i, H)$,
- $\langle \leftarrow \bar{G}, \sigma \rangle \mapsto^{k_1} \langle \Box, \sigma\mu \rangle$,
- $k = k_1 + k_2 + 1$ (and therefore $k_1, k_2 \langle k$),
- $\bar{A}' = A_1, \ldots, A_{i-1}, \bar{B}, A_{i+1}, \ldots, A_n$.

By the induction hypothesis, there exists $\bar{s}'$ such that

$$\langle \bar{G}\sigma, \bar{s}' \rangle \in lfp(T_W), \quad \text{and}$$
$$\mu \in (\mathcal{R}(Int(\bar{s}')))_{|\mathcal{V}(\bar{G}\sigma)} = (\mathcal{R}(Int(\bar{s}')))_{|\mathcal{V}(\bar{G}\sigma)}.$$

Then, by Lemma 4.14(2), we have that there exists $\bar{s}'_1$ such that

$$\langle \bar{G}, \bar{s}'_1 \rangle \in lfp(T_W), \quad \text{and}$$
$$(\sigma\mu)_{|\mathcal{D}(\sigma)} \in (\mathcal{R}(\sigma.(Int(\bar{s}'_1))))_{|\mathcal{D}(\sigma)}. \tag{2}$$

Moreover, by the induction hypothesis, there exists $\bar{s}''$ such that

$$\langle \bar{A}'\sigma\mu, \bar{s}'' \rangle \in lfp(T_W), \quad \text{and}$$
$$\vartheta' \in (\mathcal{R}(Int(\bar{s}'')))_{|\mathcal{V}(\bar{A}'\sigma\mu)} = (\mathcal{R}(Int(\bar{s}'')))_{|\mathcal{V}(\bar{A}'\sigma\mu)}. \tag{3}$$

By Lemma 4.14(2), there exists $\bar{s}''_1$ such that

$$\langle \bar{A}'\sigma\mu, \bar{s}''_1 \rangle \in lfp(T_W), \quad \text{and}$$
$$(\mathcal{R}((\sigma\mu).(Int(\bar{s}''_1))))_{|\mathcal{D}(\sigma\mu)} = (\sigma\mu\mathcal{R}(Int(\bar{s}'')))_{|\mathcal{D}(\sigma\mu)}. \tag{4}$$

Note that $\langle \bar{A}', \bar{s}_1'' \rangle \in \mathit{lfp}(T_W)$ implies the existence of a sequence of sequences of substitutions $\bar{s}'''$ such that $\langle \bar{B}, \bar{s}''' \rangle \in \mathit{lfp}(T_W)$. By definition of $T_W$, for each $s_i$ such that

$$s_i \in [\sigma.\mathit{Int}(\bar{s}_1')].\mathit{Int}(\bar{s}'''), \tag{5}$$

we have

$$\langle A_i, s_i \rangle \in \mathit{lfp}(T_W).$$

For the other atoms $A_j$ $(j \neq i)$, by equation (4), we have that there exists $s_j$ such that $\langle A_i, s_i \rangle \in \mathit{lfp}(T_W)$. Let $\bar{r} = s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n$. We have

$$\mathit{Int}(\bar{s}_1'') = \mathit{Int}(\bar{s}''', \bar{r}). \tag{6}$$

Therefore

$$\begin{aligned}
\vartheta &= (\sigma\mu\vartheta')_{|\mathcal{V}(\bar{A})} \in (\sigma\mu\mathcal{R}(\mathit{Int}(\bar{s}'')))_{|\mathcal{V}(\bar{A})} && \text{(by (3))}\\
&= (\mathcal{R}((\sigma\mu).(\mathit{Int}(\bar{s}_1''))))_{|\mathcal{V}(\bar{A})} && \text{(by (4))}\\
&\subseteq (\mathcal{R}((\mathcal{R}(\sigma.\mathit{Int}(\bar{s}_1'))).(\mathit{Int}(\bar{s}_1''))))_{|\mathcal{V}(\bar{A})} && \text{(by (2))}\\
&= (\mathcal{R}([\sigma.\mathit{Int}(\bar{s}_1')].(\mathit{Int}(\bar{s}_1''))))_{|\mathcal{V}(\bar{A})}\\
&= (\mathcal{R}(\mathit{Int}(s_1, \ldots, s_i, \ldots ; s_n)))_{|\mathcal{V}(\bar{A})}. && \text{(by (5) and (6))} \quad \square
\end{aligned}$$

The following theorem states the completeness of the operational semantics with respect to the fixpoint semantics.

**Theorem 4.17** (Completeness). *Let $W$ be a program and let $\bar{A}$ be a sequence of atoms. Then*

$$\{\vartheta \mid \exists \bar{s} \text{ locally independent on } \bar{A}\text{: } \langle \bar{A}, \bar{s} \rangle \in \mathcal{F}(W) \text{ and } \vartheta \in \mathcal{R}(\| (\bar{s}))_{|\mathcal{V}(\bar{A})}\} \subseteq \mathcal{O}_1(\leftarrow \bar{A}).$$

**Proof.** Let $\bar{s} = s_1, \ldots, s_n$. We prove the theorem by induction on the length $\#(\bar{s})$ of $\bar{s}$ (where $\#(\bar{s}) = \#(s_1) + \cdots + \#(s_n)$).

($\#(\bar{s}) = 1$): In this case, $\bar{A}$ contains only one atom, say $A$, and $\bar{s}$ contains only one substitution, say $\vartheta'$, and $\vartheta = \mathcal{R}(\vartheta'))_{|\mathcal{V}(\bar{A})} = \vartheta'_{|\mathcal{V}(\bar{A})}$. Then, there exists a clause of the form $H \leftarrow | \in W_{\vartheta(A)}$ and $\vartheta' \in \mathit{mgu}_i(A^-, H)$ holds. Then, since $\mathcal{R}(\vartheta') \neq \emptyset$, we have $\vartheta'_{|\mathcal{V}(\bar{A})} = \varepsilon$. Therefore

$$\langle \leftarrow A, \varepsilon \rangle \rightarrow^1 \langle \square, \vartheta' \rangle.$$

($\#(\bar{s}) > 1$): Let $s \in \mathit{Int}(\bar{s})$ such that $\vartheta \in (\mathcal{R}(s))_{|\mathcal{V}(\bar{A})}$. We have two cases, depending on the first element of $s$ being a critical section or not.

(1) Consider the case that there exist $\sigma$, $s'$ such that $s = \sigma.s'$ or $s = [\sigma].s'$. Assume that $\sigma$ is associate to $s_i$, i.e., $s_i = \sigma.s'_i$. Then, there exists a clause with empty guard, $H \leftarrow |\bar{B} \in W_{\mathscr{V}(A_i)}$, such that $\sigma \in mgu_i(A_i^-, H)$. Moreover, since $\mathscr{R}(s) \neq \emptyset$, we have $\sigma_{|\mathscr{V}(A_i)} = \varepsilon$. Therefore

$$\langle \leftarrow \bar{A}, \varepsilon \rangle \rightarrow^1 \langle \leftarrow (A_1, \ldots, A_{i-1}, \bar{B}, A_{i+1}, \ldots, A_n), \sigma \rangle$$

and

$$\exists \bar{r}\colon \quad \langle \bar{B}, \bar{r} \rangle \in lfp(T_W).$$

By Lemma 4.14(1), there exist $\bar{s}'$, $s''$, $\bar{s}''$, such that

- $\langle A_1\sigma, s'_1 \rangle, \ldots, \langle A_{i-1}\sigma, s'_{i-1} \rangle, \langle A_{i+1}\sigma, s'_{i+1} \rangle, \ldots, \langle A_n\sigma, s'_n \rangle, \langle \bar{B}\sigma, \bar{s}'' \rangle \in lfp(T_W)$,
- $s'' \in Int(s'_1, \ldots, s'_{i-1}, s'_{i+1}, \ldots, s'_n, \bar{s}'')$,
- $(\mathscr{R}(\sigma.s'))_{|\mathscr{V}(\bar{A}) \cup \mathscr{D}(\sigma)} = (\sigma\mathscr{R}(s''))_{|\mathscr{V}(\bar{A}) \cup \mathscr{D}(\sigma)}$.

(Note that $s' \in Int(s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n, \bar{r})$.)

(2) Consider now the case that there exist $s'$, $s''$ such that $s = [s'].s''$. Assume that $s'$ is associate to $s_i$, i.e. $s_i = [s'].s'_i$. Then there exists a clause $H \leftarrow \bar{G} | \bar{B} \in W_{\mathscr{V}(A_i)}$ such that

- $\sigma \in mgu_i(A_i^-, H)$,
- $\exists \bar{r}\colon \langle \bar{G}, \bar{r} \rangle \in lfp(T_W)$,
- $s' \in \sigma.Int(\bar{r})$.

From Lemma 4.14(1) it follows that there exists $\bar{r}'$ such that

- $\langle \bar{G}\sigma, \bar{r}' \rangle \in lfp(T_W)$, and
- $(\sigma\mathscr{R}(Int(\bar{r}')))_{|\mathscr{D}(\sigma)} = (\mathscr{R}(\sigma.Int(\bar{r})))_{|\mathscr{D}(\sigma)}$.

By the induction hypothesis we have for each $r' \in Int(\bar{r}')$,

$$\langle \leftarrow \bar{G}, \sigma \rangle \rightarrow^k \langle \square, \sigma\tau \rangle$$

(for an appropriate $k$) where $\tau_{|\mathscr{V}(\bar{G}\sigma)} \in (\mathscr{R}(r'))_{|\mathscr{V}(\bar{G}\sigma)}$. Moreover, since

$$(\mathscr{R}(s'))_{|\mathscr{V}(\bar{G}\sigma)} \subseteq (\mathscr{R}([\sigma.Int(\bar{r})]))_{|\mathscr{V}(\bar{G}\sigma)} = (\sigma.\mathscr{R}(Int(\bar{r}')))_{|\mathscr{D}(\sigma)},$$

and $\mathscr{R}(s')_{|\mathscr{V}(\bar{G}\sigma)} \neq \emptyset$, we have that $\sigma$ and $\tau$ do not instantiate variables of $A_i$. Therefore

$$\langle \leftarrow \bar{A}, \varepsilon \rangle \rightarrow^{k+1} \langle \leftarrow (A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n, \bar{B}), \sigma\tau \rangle.$$

The rest follows as in case (1). $\square$

**Example 4.18.** (1) Consider the program $\{p(y) \leftarrow q(y)|., q(a) \leftarrow |.\}$, with declaration-mode $\{p(?), q(\hat{\ })\}$, and consider the goal $\leftarrow p(x)$. The possible $s$'s such that
$\langle p(x), s \rangle \in lfp(T)$, are those of the form $s = [\{y/x^-\}.\{y/a\}]_{\{x^-\}}$. We have

$$\mathscr{R}(s) = disann_{\{x^-\}}(\mathscr{E}(\{y/x^-\} \mathbin{\hat{\circ}} \{y/a\}))$$
$$= disann_{\{x^-\}}(\mathscr{E}(\{\{x^-/a, y/a\}\})) = \emptyset,$$

and indeed no refutations are possible.

(2) Consider now the program $\{p(y) \leftarrow | q(y).,\ q(a) \leftarrow |.\}$, with the same declaration mode. The possible $s$'s are of the form $s = [\{y/x^-\}]_{\{x^-\}}.\{y/a\}$. We have

$$\begin{aligned}\mathscr{R}(s) &= \mathscr{R}(disann_{\{x^-\}}(\mathscr{E}(\{y/x^-\})) \hat{\circ} \{y/a\}) \\ &= \mathscr{R}(\{y/x\} \hat{\circ} \{y/a\}) = \{\{x/a, y/a\}\},\end{aligned}$$

and we notice that indeed there exists a refutation for $\leftarrow p(x)$ giving the answer $\{x/a\}$.

Now we consider again the example showed in the introduction (*deadlock* situation), which illustrates the necessity to use streams-like structures.

**Example 4.19.** (1) Consider the program $\{p(a, b) \leftarrow |.,\ q(a, b) \leftarrow |.\}$, with declaration-mode $\{p(?, \hat{}\ ), q(\hat{}\ , ?)\}$, and consider the goal $\leftarrow p(x, y), q(x, y)$. We have

$$\langle p(x, y), s_1\rangle, \langle q(x, y), s_2\rangle \in lfp(T),$$

for $s_1 = [\{x^-/a\}]_{\{x^-\}}.\{y/b\}$ and $s_2 = [\{y^-/b\}]_{\{y^-\}}.\{x/a\}$. For all the possible interleavings $s \in s_1 \| s_2$, we get $\mathscr{R}(s) = \emptyset$. Indeed, no refutations are possible (*deadlock*).

(2) Consider now the program $\{p(z, b) \leftarrow | r(z).,\ r(a) \leftarrow |.,\ q(a, b) \leftarrow |.\}$, with the same declaration-mode for $p$ and $q$, and with $r(?)$. We have

$$\langle p(x, y), s_1\rangle, \langle q(x, y), s_2 > \in lfp(T),$$

for $s_1 = [\{z/x^-\}]_{\{x^-\}}.\{y/b\}.[\{z^-/a\}]_{\{z^-\}}$ and $s_2 = [\{y^-/b\}]_{\{y^-\}}.\{x/a\}$. We have

$$s = [\{z/x^-\}]_{\{s^-\}}.\{y/b\}.[\{y^-/b\}]_{\{y^-\}}.\{x/a\}.[\{z^-/a\}]_{\{z^-\}} \in s_1 \| s_2 \quad \text{and}$$

$$\{x/a, y/b, z/a\} \in \mathscr{R}(s).$$

Indeed, there exists a refutation of the goal $\leftarrow p(x, y), q(x, y)$ giving the answer $\{x/a, y/b\}$.

## 5. Denotational semantics

The semantic universe $M_2$ of the operational semantics offers too little structure to define a compositional semantics. One of the reasons is that it is too coarse to distinguish between different kinds of deadlock. A standard solution stemming from the semantic studies of imperative languages is to use tree-like structures. Following [9], we introduce a so-called reflexive domain, which is a complete metric space obtained as the (unique) solution of a reflexive domain equation. (We omit the proof of its existence; in [9] and [2], it is solved in general domain equations in a metric setting.)

**Definition 5.1.** The set $(p, q \in)\ P$ is given as the unique complete metric space satisfying

$$P \cong \{p_0\} \cup \mathscr{P}_c(\Gamma \times P).$$

where $\cong$ means "is isometric to" and $\mathcal{P}_c(\Gamma \times P)$ denotes the set of all *closed* subsets of $\Gamma \times P$. Further $\Gamma$ is given by $(\alpha \in)\ \Gamma = V \cup V^{[\,]}$ with

$$(f \in)\ V = Subst \rightarrow Subst_\delta, \qquad V^{[\,]} = \{[f]: f \in V\}.$$

Here $Subst_\delta = Subst \cup \{\delta\}$ and $\delta$ is a special element denoting deadlock.

Elements of $P$ are called *processes.* A process $p$ can either be $p_0$, which stands for termination, or a closed set $\{\langle \alpha_i, p_i \rangle: i \in I\}$, for some index set $I$. In that case, $p$ has the choice among the steps $\langle \alpha_i, p_i \rangle$. Each step consists of some action $\alpha_i$, which is a state transformation, and a *resumption* $p_i$ of this action, that is, the remaining actions to be taken after this action.

The main difference between $P$ and $M_2$, as was already observed above, is the fact that $P$ contains tree-like structures whereas $M_2$ is a set of (subsets of) streams. In addition, there are two other important differences. First, we use state transforming functions rather that states (substitutions). This functionality is mandatory if we want to define a compositional semantics. Secondly, *internal* steps are visible in $P$, which is not the case in the operational semantics. For this purpose we distinguish between two kinds of actions: an element $f \in V$ represents an internal computation step, which corresponds to a step in the evaluation of a guard. An action $[f] \in V^{[\,]}$ indicates an external step or to be more precise, the end of an internal computation. (This implies that external steps are modelled as internal computations of length 1.) A typical example of a process is

$$p = \{\langle f_1, \{\langle [f_2], \{\langle [f_3], p_0 \rangle\}\rangle,$$
$$\langle f_4, \{\langle [f_5], p_0, \langle [f_6], p_0 \rangle\}\rangle\}\rangle\}.$$

We shall use the following semantic operators.

**Definition 5.2.** We define $;, \| : P \times P \rightarrow P$ and $\mathrm{int}: P \rightarrow P$:

(1) $p_0;q = q,$

$p;q = \{\langle \alpha, p';q \rangle \mid \langle \alpha, p' \rangle \in p\}.;q = \{\langle \alpha, p';q \rangle \mid \langle \alpha, p' \rangle \in p\}.$

(2) $p_0 \| q = q \| p_0 = q,$

$p \| q = p \mathbin{\|\!\!\lfloor} q \cup q \mathbin{\|\!\!\lfloor} p,$

$p \mathbin{\|\!\!\lfloor} q = \{\langle \alpha, p' \rangle \mathbin{\|\!\!\lfloor} q \mid \langle \alpha, p' \rangle \in p\},$

$\langle f, p' \rangle \mathbin{\|\!\!\lfloor} q = \langle f, p' \mathbin{\|\!\!\lfloor} q \rangle,\ \langle [f], p' \rangle \mathbin{\|\!\!\lfloor} q = \langle [f], p' \| q \rangle.$

(3) $int(p_0) = p_0,$

$int(p) = \{\langle f, int(p') \rangle \mid (\langle f, p' \rangle \in p \vee \langle [f], p' \rangle \in p) \wedge p' \neq p_0\}$

$\cup \{\langle [f], p_0 \rangle \mid \langle f, p_0 \rangle \in p \vee \langle [f], p_0 \rangle \in p\}.$

These definitions are recursive and can be given in a formally correct way by defining every operator as the unique fixed point of a suitably defined contraction.

The definition of; is straightforward. The parallel merge operator $\|$ models the parallel execution of two processes by the interleaving of their respective steps. In determining all possible interleavings, the notions of internal and external steps are crucial; inside an internal computation, no interleaving with other processes is allowed. Only after the last internal step, indicated by the brackets [ ], do we have an interleaving point. This explains the definitions of the (auxiliary) operator for the *left merge*, which is like the ordinary merge but which always starts with a step from the left process. If this step is internal (but not the last step of the internal computation) then we have to continue with a next step of this left process: $\langle f, p'\rangle \mathbin{|\!\lfloor} q = \langle f, p' \mathbin{|\!\lfloor} q\rangle$. If, on the other hand, an interleaving point is reached then we switch back to the ordinary merge again: $\langle [f], p'\rangle \mathbin{|\!\lfloor} q = \langle [f], p' \| q\rangle$.

The operator *int* makes a computation *internal* by removing all internal interleaving points. This implies that in the parallel composition $int(p) \| q$ (for two arbitrary processes $p$ and $q$) none of the paths in $p$ will be interleaved with any step of $q$.

Now we are already for the definition of the denotational semantics. Let $W$ be a fixed program.

**Definition 5.3.** We define $\mathscr{D} : \mathscr{P}(\mathit{Var}) \to \mathit{Goal} \to P$ as follows:

(1) $\mathscr{D}[\![X]\!][\![\square]\!] = p_0$.

(2) $$\mathscr{D}[\![X]\!][\![\leftarrow A]\!] = \bigcup \{\mathit{int}(\{\langle f_i(A, H, X), \mathscr{D}[\![X \cup \mathit{invar}(A)]\!][\![\leftarrow \bar{G}]\!]\rangle\}); \\ (\mathscr{D}[\![X]\!][\![\mathit{outunif}(A, H), \bar{B}]\!]) : H \leftarrow \bar{G} | \bar{B} \in W\}$$

with

$$f_i(A, H, X) = \lambda\vartheta. \begin{cases} \vartheta\, \mathit{mgu}_i(A\vartheta, H) & \text{if } \mathit{mgu}_i(A\vartheta, H)\downarrow \text{ and } \mathit{mgu}_i(A\vartheta, H)|_{X\vartheta \cup \mathit{invar}(A\vartheta)} = \varepsilon, \\ \delta & \text{otherwise.} \end{cases}$$

(Here $X\vartheta = \bigcup \{\mathit{var}(\vartheta(x)) : x \in X\}$.)

(3) $\mathscr{D}[\![X]\!][\![\leftarrow \mathit{outunif}(A, H)]\!] = \{\langle f_0(A, H, X), p_0\rangle\}$

with

$$f_0(A, H, X) = \lambda\vartheta. \begin{cases} \vartheta\, \mathit{mgu}_0(A\vartheta, H) & \text{if } \mathit{mgu}_0(A\vartheta, H)\downarrow, \\ \delta & \text{otherwise.} \end{cases}$$

(4) $\mathscr{D}[\![X]\!][\![\leftarrow \bar{A}, \bar{B}]\!] = \mathscr{D}[\![X]\!][\![\leftarrow \bar{A}]\!] \| \mathscr{D}[\![X]\!][\![\leftarrow \bar{B}]\!]$.

(Here the notation $\downarrow$ is used to express the non-existence of the most general unifier.)

(Note that the definition of $\mathscr{D}$ is recursive; like the semantic operators, it can be given as the fixed point of a contraction.) The first argument $X$ of $\mathscr{D}$ indicates the set of variables that are not to be bound during the computation of the goal at hand (i.e., the second argument of $\mathscr{D}$). It is used in the definitions of $f_i$ and $f_0$. In clause

(2), $X$ is changed into $X \cup \mathit{invar}(A)$, because a new guard computation is entered there. The set of variables $X$ that are not allowed to be bound (stemming from the computation sofar) is extended with the set $\mathit{invar}(A)$ of the input variables occurring in $A$, because in the computation of the guard $\bar{G}$ these should not get bound. After the computation of $\bar{G}$, the variables that are not to be bound are put to $X$ again, thus indicating that the input variables of $A$ may be bound again. In clause (2) we further have that the computation of the unification and the guard is made internal by an application of the function *int*, since it should not be interleaved with other (guard) computations that may run in parallel.

## 6. Correctness of $\mathscr{D}$ with respect to $\mathcal{O}_2$

We shall relate $\mathcal{O}_2$ and $\mathscr{D}$ via a function $\mathit{yield}: P \to \mathit{Subst} \to M_2$. For technical convenience we shall slightly adapt the definition of $\mathcal{O}_2$ by allowing the computation of a goal to start with an arbitrary substitution, not necessarily the empty one. Moreover we shall define this adapted version of $\mathcal{O}_2$ as the fixed point of a contraction $\Phi$, which will allow for an easy equivalence proof. First we turn $M_2$ into a complete metric space.

**Definition 6.1.** We define $M_2 = \mathscr{P}_c(\mathit{Subst}_\delta^\infty)$, where $\mathscr{P}_c$ denotes the set of all *closed* subsets. The set $M_2$ is a complete metric space if we supply it with the Hausdorff metric induced by the usual metric on $\mathit{Subst}_\delta^\infty$.

Now we can define a contraction

$$\Phi : (\mathit{Goal} \to \mathit{Subst} \to M_2) \to (\mathit{Goal} \to \mathit{Subst} \to M_2)$$

by

$$\Phi(F)[\![\leftarrow \bar{A}]\!](\vartheta) = \bigcup \{\vartheta'.F[\![\leftarrow \bar{A}']\!](\vartheta') : \langle \leftarrow \bar{A}, \vartheta\rangle \to \langle \leftarrow \bar{A}', \vartheta'\rangle\}$$

if this set is non-empty, and by

$$\Phi(F)[\![\leftarrow \bar{A}]\!](\vartheta) = \{\delta\}$$

otherwise. Note that the complete metric on $M_2$ induces a complete metric on $\mathit{Goal} \to \mathit{Subst} \to M_2$ in the usual way. Next we use Banach's Theorem, which says that a contraction on a complete metric space has a unique fixed point. So we put

$$\mathcal{O}_2 = \text{fixed-point}(\Phi).$$

The function *yield* is defined as follows.

**Definition 6.2.** Let the function $\mathit{yield}: P \to \mathit{Subst} \to M_2$ be given by

$$\mathit{yield}(p_0)(\vartheta) = \{e\} \quad \text{(the empty sequence)},$$

$$\mathit{yield}(p)(\vartheta) = \bigcup_\delta \{\vartheta'.\mathit{yield}(p_n)(\vartheta') : \langle f_1, p_1\rangle \in p \wedge \cdots \wedge \langle f_{n-1}, p_{n-1}\rangle \in p_{n-2} \wedge \langle [f_n], p_n\rangle \in p_{n-1} \wedge (f_n \circ \cdots \circ f_1)(\vartheta) = \vartheta'\}.$$

(The attentive reader might observe that the function *yield* is not well defined, because in general $yield(p)(\vartheta)$ is not closed. He is right. Happily, however, we are saved by the observation that the restriction of *yield* to the set $\{p: \exists \bar{A}, X(p = \mathscr{D}[\![X]\!][\![\leftarrow\bar{A}]\!])\}$ always delivers closed sets. This turns out to be everything we need. We leave the details to the above-mentioned reader.)

In the above definition the operation $\bigcup_\delta$ is used. It is defined by

$$\bigcup_{\delta} X = \begin{cases} \bigcup X\backslash\{\delta\} & \text{if } \bigcup X\backslash\{\delta\} \neq \emptyset, \\ \{\delta\} & \text{otherwise.} \end{cases}$$

The function *yield* performs four abstractions at the same time. First, it turns a process (a tree-like structure) into a set of streams. Second, it computes for every initial state $\vartheta$ and state transformation $f$ the next state by applying $f$ to $\vartheta$. This result is then passed through to the next state transformation in the process. Third, *yield* performs the function composition of all functions occurring in a sequence $f_1, \ldots, f_n$ that is induced by a finite path in $p$ like

$$\langle f_1, p_1\rangle, \ldots, \langle f_{n-1}, p_{n-1}\rangle, \langle [f_n], p_n\rangle.$$

Such a sequence represents an internal computation, the end of which is indicated by $[f_n]$. Finally, the function *yield* removes all infinite internal computations.

Now we are ready to prove the equivalence of the denotational semantics $\mathscr{D}$ and the operational semantics $\mathcal{O}_2$. In the theorem below we shall allow ourselves the following abuse of language by writing $yield \circ \mathscr{D}$ for

$$\lambda\vartheta.\, \lambda \leftarrow \bar{A}.\, yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow\bar{A}]\!])(\vartheta).$$

**Theorem 6.3.** $\mathcal{O}_2 = yield \circ \mathscr{D}$.

**Proof.** We show that for all $\vartheta$, $\vartheta'$, $\bar{A}$, $\bar{A}'$,

$$\langle\leftarrow\bar{A}, \vartheta\rangle \rightarrow \langle\leftarrow\bar{A}', \vartheta'\rangle$$
$$\Leftrightarrow\ yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow\bar{A}]\!])(\vartheta) \supseteq \vartheta' \cdot yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow\bar{A}']\!])(\vartheta'). \tag{1}$$

From this it follows that

$$\Phi(yield \circ \mathscr{D})[\![\leftarrow\bar{A}]\!](\vartheta) = yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow\bar{A}]\!])(\vartheta) \tag{2}$$

since

$$\begin{aligned} &\Phi(yield \circ \mathscr{D})[\![\leftarrow\bar{A}]\!](\vartheta) \\ &= \bigcup_{\delta}\{\vartheta' \cdot yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow\bar{A}']\!])(\vartheta') : \langle\leftarrow\bar{A}, \vartheta\rangle \rightarrow \langle\leftarrow\bar{A}', \vartheta'\rangle\} \quad \text{(definition } \Phi) \\ &\subseteq yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow\bar{A}]\!])(\vartheta) \quad \text{by (1)} \end{aligned}$$

and

$$\begin{aligned} &yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow\bar{A}]\!])(\vartheta) \\ &\subseteq \bigcup_{\delta}\{\vartheta' \cdot yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow\bar{A}']\!])(\vartheta') : \langle\leftarrow\bar{A}, \vartheta\rangle \rightarrow \langle\leftarrow\bar{A}', \vartheta'\rangle\}. \end{aligned}$$

The latter inclusion holds by (1) and the fact that

$$\vartheta' \cdot \mathit{yield}(p)(\vartheta') \subseteq \mathit{yield}(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!])(\vartheta) \Rightarrow \exists \leftarrow \bar{A}' : p = \mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}']\!], \tag{3}$$

which is straightforward from the definitions of $\mathcal{D}$ and *yield*. Now the theorem follows from (2) since it states that apart from $\mathcal{O}_2$ also $\mathit{yield} \circ \mathcal{D}$ is a fixed point of $\Phi$. Both fixed points have to be equal by Banach's Theorem.

So let us prove (1). We distinguish between four cases.

*Case* 1: $\square$, trivial.

*Case* 2: $\leftarrow A$. By definition of $\rightarrow$ we have $\langle \leftarrow A, \vartheta \rangle \rightarrow \langle \leftarrow \bar{A}, \vartheta' \rangle$ if and only if there exist $H \leftarrow \bar{G} | \bar{B}$ and $\bar{\vartheta}$ such that

$$\langle \leftarrow \bar{G}, \mathit{mgu}_i(A\vartheta, H) \rangle \xrightarrow{*} \langle \square, \bar{\vartheta} \rangle \tag{4}$$

and

$$\bar{\vartheta}|_{\mathit{invar}(A)} = \varepsilon \wedge \vartheta' = \vartheta\bar{\vartheta} \wedge \leftarrow \bar{A} = \leftarrow \mathit{outunif}(A, H), \bar{B}.$$

We use induction on the depth of proof trees of transitions and observe that every transition in the sequence $\xrightarrow{*}$ in (4) has a degree that is strictly less that that of $\langle \leftarrow A, \vartheta \rangle \rightarrow \langle \leftarrow \bar{A}, \vartheta' \rangle$. It follows from the induction hypothesis applied to every one of these transitions in $\xrightarrow{*}$ that there exist $n \geq 0$ and substitutions $\vartheta_1, \ldots, \vartheta_n$ such that

$$\vartheta_1 \cdot \ldots \cdot \vartheta_n \cdot \bar{\vartheta} \cdot \mathit{yield}(\mathcal{D}[\![\emptyset]\!][\![\square]\!])(\bar{\vartheta}) \subseteq \mathit{yield}(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{G}]\!])(\mathit{mgu}_i(A\vartheta, H))$$

or, since $\mathit{yield}(\mathcal{D}[\![\emptyset]\!][\![\square]\!])(\mathit{mgu}_i(A\vartheta, H)) = \{e\}$,

$$\vartheta_1 \cdot \ldots \cdot \vartheta_n \cdot \bar{\vartheta} \in \mathit{yield}(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{G}]\!])(\mathit{mgu}_i(A\vartheta, H)).$$

Now we have

$$\vartheta_1 \cdot \ldots \cdot \vartheta_n \cdot \bar{\vartheta} \in \mathit{yield}(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{G}]\!])(\mathit{mgu}_i(A\vartheta, H))$$

$\Leftrightarrow\ \bar{\vartheta} \in \mathit{yield}(\mathit{int}(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{G}]\!]))(\mathit{mgu}_i(A\vartheta, H))$

(definitions *yield* and *int*)

$\Leftrightarrow\ \bar{\vartheta} \in \mathit{yield}(\mathit{int}(\mathcal{D}[\![\mathit{invar}(A)]\!][\![\leftarrow \bar{G}]\!]))(\mathit{mgu}_i(A\vartheta, H))$

(using that $\bar{\vartheta}|_{\mathit{invar}(A)} = \varepsilon$)

$\Leftrightarrow\ \bar{\vartheta} \in \mathit{yield}(\mathit{int}(\mathcal{D}[\![\mathit{invar}(A)]\!][\![\leftarrow \bar{G}]\!]))(\vartheta\, \mathit{mgu}_i(A\vartheta, H))$

$\Leftrightarrow\ \vartheta' \in \mathit{yield}(\mathit{int}(\{\langle f_i(A, H, \emptyset), \mathcal{D}[\![\mathit{invar}(A)]\!][\![\leftarrow \bar{G}]\!]\rangle\}))(\vartheta).$

(definitions *yield*, $f_i(A, H, \emptyset)$; recall that $\vartheta' = \vartheta\bar{\vartheta}$)

From the definition of $\mathcal{D}$ we have

$$\begin{aligned} &\mathit{yield}(\mathcal{D}[\![\emptyset]\!][\![\leftarrow A]\!])(\vartheta) \\ &\quad = \mathit{yield}(\mathit{int}(\{\langle f_i(A, H, \emptyset), \mathcal{D}[\![\mathit{invar}(A)]\!][\![\leftarrow \bar{G}]\!]\rangle\}); \\ &\qquad\qquad \mathcal{D}[\![\emptyset]\!][\![\leftarrow \mathit{outunif}(A, H), \bar{B}]\!] : H \leftarrow \bar{G} | \bar{B} \in W\})(\vartheta) \\ &\quad = \bigcup \{\vartheta' \cdot \mathit{yield}(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \mathit{outunif}(A, H), \bar{B}]\!])(\vartheta') : \\ &\qquad \vartheta' \in \mathit{yield}(\mathit{int}(\{\langle f_i(A, H, \emptyset), \mathcal{D}[\![\mathit{invar}(A)]\!][\![\leftarrow \bar{G}]\!]\rangle\}))(\vartheta), \\ &\qquad H \leftarrow \bar{G} | \bar{B} \in W\}. \end{aligned}$$

(For the latter equality we use the fact that

$$yield(int(p);\, q)(\vartheta) = \bigcup_{\delta} \{\vartheta' \cdot yield(q)(\vartheta') \colon \vartheta' \in yield(int(p))(\vartheta)\}$$

for all $p$, $q$ and $\vartheta$, which is straightforward from the definitions of *yield* and *int*.) Using this characterization of $yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow A]\!])(\vartheta)$ we can conclude that (5) above is equivalent with

$$\vartheta' \cdot yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow outunif(A, H), \bar{B}]\!])(\vartheta') \subseteq yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow A]\!])(\vartheta).$$

Summarizing we have

$$\langle \leftarrow A, \vartheta\rangle \to \langle (\leftarrow outunif(A, H), \bar{B}), \vartheta'\rangle$$
$$\vartheta' \cdot yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow outunif(A, H), \bar{B}]\!])(\vartheta') \subseteq yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow A]\!])(\vartheta),$$

which is what we wanted to show.

*Case* 3: $\leftarrow outunif(A, H)$, trivial.

*Case* 4: $\leftarrow \bar{A}, \bar{B}$. We have $\langle \leftarrow \bar{A}, \bar{B}, \vartheta\rangle \to \langle \leftarrow \bar{A}'', \vartheta'\rangle$ if and only if

$$\langle \leftarrow \bar{A}, \vartheta\rangle \to \langle \leftarrow \bar{A}', \vartheta'\rangle, \qquad \leftarrow \bar{A}'' = \leftarrow \bar{A}', \bar{B}$$

or

$$\langle \leftarrow \bar{B}, \vartheta\rangle \to \langle \leftarrow \bar{B}', \vartheta'\rangle, \qquad \leftarrow \bar{A}'' = \leftarrow \bar{A}, \bar{B}'.$$

We consider only the first case, the second being almost identical. By induction, again to the depth of the proof tree for this transition, we have

$$\langle \leftarrow \bar{A}, \vartheta\rangle \to \langle \to \bar{A}', \vartheta'\rangle \Leftrightarrow yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!])(\vartheta) \supseteq yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}']\!])(\vartheta'). \tag{6}$$

From the definition of $\mathscr{D}$ and *yield* it follows that

$$\begin{aligned} yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}, \bar{B}]\!])(\vartheta) &= yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!] \parallel \mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{B}]\!])(\vartheta) \\ &\supseteq yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!] \mathbin{\lfloor\!\lfloor} \mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{B}]\!])(\vartheta) \\ &= \bigcup_{\delta} \{\vartheta' \cdot yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}']\!] \parallel \mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{B}]\!])(\vartheta) \colon \\ &\qquad \vartheta' \cdot yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}']\!])(\vartheta') \subseteq yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!])(\vartheta)\} \\ &= \bigcup_{\delta} \{\vartheta' \cdot yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}', \bar{B}]\!])(\vartheta) \colon \\ &\qquad \vartheta' \cdot yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}']\!])(\vartheta') \subseteq yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!])(\vartheta)\}. \end{aligned}$$

(The last but one equality follows from (3) and the observation that

$$yield(p \mathbin{\lfloor\!\lfloor} q)(\vartheta) = \bigcup_{\delta} \{\vartheta' \cdot yield(p' \parallel q)(\vartheta') \colon \vartheta' \cdot yield(p')(\vartheta') \subseteq yield(p)(\vartheta)\}$$

for all $p$, $q$ and $\vartheta$.) Thus we see that (6) is equivalent with

$$\vartheta' \cdot yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}', \bar{B}]\!])(\vartheta') \subseteq yield(\mathscr{D}[\![\emptyset]\!][\![\leftarrow \bar{A}, \bar{B}]\!])(\vartheta).$$

This concludes the proof of Case 4. □

## 7. Conclusions and future work

We have defined a declarative semantics that fully models (i.e., it is equivalent to) the Success Set of a set of concurrent logic programming languages with input-constraints and commit operator. Then, we have defined a denotational semantics, correct with respect to the operational one, that (compositionally) also models the finite failures and the infinite computations. Since we have abstracted from the particulars of any specific language, our semantic descriptions apply to various instances of concurrent logic languages, such as PARLOG, Guarded Horn Clauses and Concurrent Prolog.

If we compare the denotational semantics given here to the ones given in [6, 7] for Concurrent Prolog and Guarded Horn Clauses, respectively, we observe that it is more abstract, that is, makes less distinctions. Moreover, it is in some sense closer to the declarative model (than in the case of [7]), because the restrictions imposed on unifications by the input constraints for a program are treated in the same way in both the denotational and the declarative model.

Still, the denotational model is not fully abstract and the construction of such a model remains a topic for further research. Another question still to be investigated is the relation between the denotational and the declarative semantics. Here both models are related via their corresponding operational semantics, but it would be interesting to formalize their relationship more directly. In [8], a direct equivalence is established for HCL.

## 8. Appendix: the extended unification algorithm

We give an extended version of the unification algorithm based on the one presented in [1], that works on finite sets of pairs. Given a finite set of finite sets of terms $M$, consider the (finite) set of pairs.

$$M_{pairs} = \bigcup_{S \in M} \{\langle t, u\rangle \mid t, u \in S\}.$$

We define the unifiers of a set $\{\langle t_1, u_1\rangle, \ldots, \langle t_n, u_n\rangle\}$ as the ones of $\{\{t_1, u_1\}, \ldots, \{t_n, u_n\}\}$. Of course, $M$ and $M_{pairs}$ are equivalent (i.e., they have the same unifiers). A set of pairs is called *solved* if it is of the form

$$\{\langle v_1, t_1\rangle, \ldots, \langle v_n, t_n\rangle\}$$

where all the $v_i$'s are distinct elements of $Var \cup Var^-$, $v_i \notin \mathcal{V}(t_1, \ldots, t_n)$, and, if $v_i \in Var$ and $t_i \neq v_i^-$, then $v_i^- \notin \mathcal{V}(v_1, \ldots, v_n, t_1, \ldots, t_n)$. For $P$ solved, define $\gamma_P = \{v_1/t_1, \ldots, v_n/t_n\}$, and $\delta_P = \gamma_P\gamma_P$.

The following algorithm transforms a set of pairs into an equivalent one which is solved, or halts with failure if the set has no unifiers.

**Definition 8.1** (*Extended unification algorithm*). Let $P$, $P'$ be sets of pairs. Define $P \Rightarrow P'$ if $P'$ is obtained from $P$ by choosing in $P$ a pair of the form below and by performing the corresponding action

| | | | |
|---|---|---|---|
| (1) | $\langle f(t_1, \ldots, t_n), f(u_1, \ldots, u_n)\rangle$ | | replace by the pairs $\langle t_1, u_1\rangle, \ldots, \langle t_n, u_n\rangle$ |
| (2) | $\langle f(t_1, \ldots, t_n), g(u_1, \ldots, u_n)\rangle$ where $f \neq g$ | | halt with failure |
| (3) | $\langle v, v\rangle$ | where $v \in Var \cup Var^-$ | delete the pair |
| (4) | $\langle t, v\rangle$ | where $v \in Var \cup Var^-$, $t \notin Var \cup Var^-$ | replace by the pair $\langle v, t\rangle$ |
| (5) | $\langle x, t\rangle$ | where $x \in Var$, $x \neq t$, $x^- \neq t$, $x$ or $x^-$ occurs in other pairs, or $x \in \mathcal{V}(t)$ or $x^- \in \mathcal{V}(t)$ | if $x \in \mathcal{V}(t)$ or $x^- \in \mathcal{V}(t)$ then halt with failure, otherwise apply the substitution $\{x/t\}$ to all the other pairs, |
| (6) | $\langle x, x^-\rangle$ | where $x \in Var$, and $x$ occurs in other pairs | apply the substitution $\{x/x^-\}$ to all the other pairs, |
| (7) | $\langle x^-, t\rangle$ | where $x^- \in Var^-$, $x^- \neq t$ and $x^-$ occurs in other pairs, or $x^- \in \mathcal{V}(t)$ | if $x^- \in \mathcal{V}(t)$ then halt with failure, otherwise apply the substitution $\{x^-/t\}$ to all the other pairs. |

We will write $P \Rightarrow fail$ if a failure is detected (steps 2, 5 or 7).

Let $\Rightarrow^*$ be the reflexive-transitive closure of the relation $\Rightarrow$, and let $P_{sol}$ be the set $P_{sol} = \{P' \mid symm(P) \Rightarrow^* P'$, and $P'$ is solved$\}$, where

$$symm(\{\langle t_1, u_1\rangle, \ldots, \langle t_n, u_n\rangle\}) = \{\langle t_1, u_1\rangle, \ldots, \langle t_n, u_n\rangle\} \cup \{\langle t_1^-, u_1^-\rangle, \ldots, \langle t_n^-, u_n^-\rangle\}.$$

The set of substitutions determined by the algorithm is

$$\Delta(P) = \{\delta_{P'} \mid P' \in P_{sol}\}.$$

The following proposition shows that the set of the idempotent most general unifiers of $M$ is finite and can be computed in finite time by the extended unification algorithm.

**Proposition 8.2.** *Let P be a finite set of pairs, and M be a finite set of finite sets of terms.*

(1) (finiteness) *The relation* $\Rightarrow$ *is* finitely branching *and* noetherian (*i.e.*, terminating).

(2) (solved form) *If P is in* normal form (*i.e.*, *if there exist no P′ such that* $P \Rightarrow P'$), *then P is in solved form.*

(3) (soundness) $\Delta(P) \subseteq mgu(P)$.

(4) (completeness) $mgu(M) \subseteq \Delta(M_{pairs})$.

(5) $P \Rightarrow^*$ fail *iff P is not unifiable.*

**Proof.** (1) (*finiteness*) By definition, $\Rightarrow$ is finitely branching iff for each $P$ there is only a finite number of $P'$ such that $P \Rightarrow P'$. At each step, the number of choices in the algorithm is bound by the number of pairs in the current set. Therefore, in order to show that the $\Rightarrow$ is finitely branching upon the elements of $\{P' \mid P \Rightarrow^* P'\}$ (for $P$ finite) it is sufficient to prove that each $P'$ derived from $P$ has a finite number of pairs. This follows from the fact that $\Rightarrow$ preserves finiteness; in fact only step (1) can increase the number of pairs, and it can add, each time, only a finite number of them.

By definition, $\Rightarrow$ is noetherian iff there are no infinite sequences $P_1 \Rightarrow P_2 \Rightarrow \cdots P_n \Rightarrow \cdots$. In order to show that $\Rightarrow$ is noetherian on the sets derived from a finite set $P$, it is sufficient to note that:

- For each variable in the original set $P$, steps (5), (6) and (7) can be performed at most once. Therefore they can be performed only a finite number of times.
- Steps (1) and (4) strictly diminish the number of occurrences of function symbols at the left hand side of the equations. Therefore (when steps (5), (6) and (7) cannot be performed anymore) they can be performed only finitely many times.
- In absence of step (1), step (3) can be applied only a finite number of times.
- Step (2) can be performed only once.

(2) (*solved form*) The unapplicability of steps (1), (2) and (4) ensures that condition (1) is satisfied. Since steps (5), (6) and (7) are not performable, conditions (2) and (3) hold. Finally, also condition (4) is implied by the unapplicability of step (5).

(3) (*soundness*) In order to prove the soundness of the algorithm we need the following lemma.

**Lemma 8.3.** *Let P be a set of pairs. Let $P' \in P_{sol}$. Then $\delta_{P'} \in mgu(P')$.*

**Proof.** Let $P' = \{\langle v_1, t_1\rangle, \ldots, \langle v_n, t_n\rangle\} \in P_{sol}$. Then, for any $v \in Var \cup Var^-$, we have three cases:

(a) $v = v_i$, for a given $i$ $(1 \leq i \leq n)$. In this case, $v\delta_{P'} = v\gamma_{P'}\gamma_{P'} = t_i\gamma_{P'} =$ (since $P'$ is in solved form) $= t_i$.

(b) $v = v_i^-$, for a given $i$ $(1 \leq i \leq n)$. In this case $v\delta_{P'} = v\gamma_{P'}\gamma_{P'} = t_i^-\gamma_{P'}$.

(c) $v \neq v_i$, $v \neq v_i^-$, for all $i = 1, \ldots, n$. In this case $v\delta_{P'} = v\gamma_{P'}\gamma_{P'} = v\gamma_{P'} = v$.

(*idempotency*) We have to show that for any $v \in Var \cup Var^-$, $v\delta_{P'}\delta_{P'} = v\delta_{P'}$.

(a) If $v = v_i$, for a given $i$ $(1 \leq i \leq n)$, then $v\delta_{P'}\delta_{P'} = t_i\delta_{P'} = v\gamma_{P'}\gamma_{P'} =$ (since $P'$ is in solved form) $= t_i = v\delta_{P'}$.

(b) If $v = v_i^-$, for a given $i$ $(1 \leq i \leq n)$, then $v\delta_{P'}\delta_{P'} = t_i^-\gamma_{P'}\gamma_{P'} =$ (since $P'$ is in solved form) $= t_i^-\gamma_{P'} = v\delta_{P'}$.

(c) If $v \neq v_i$, $v \neq v_i^-$, for all $i = 1, \ldots, n$, then $v\delta_{P'}\delta_{P'} = v\delta_{P'}(= v)$.

(*unifier*) For each $i = 1, \ldots, n$, we have $v_i\delta_{P'} = t_i =$ (since $P'$ is in solved form) $= t_i\delta_{P'}$. Moreover, $v_i^-\delta_{P'} = t_i^-\gamma_{P'} =$ (since $P'$ is in solved form) $= t_i^-\gamma_{P'}\gamma_{P'} = t_i^-\delta_{P'}$.

(*most general*) Let $\sigma$ be a unifier of $P'$. We show that $\delta_{P'}\sigma = \sigma$.

(a) If $v = v_i$, for a given $i$ $(1 \leq i \leq n)$, then $v\delta_{P'}\sigma = t_i\sigma =$ (since $\sigma$ is a unifier of $P'$) $= v_i\sigma = v\sigma$.

(b) If $v = v_i^-$, for a given $i$ $(1 \leq i \leq n)$, then $v\delta_{P'}\sigma = t_i^-\gamma_{P'}\sigma =$ (since $\sigma$ is a unifier of $P'$) $= t_i^-\sigma =$ (since $\sigma$ is a unifier of $P'$) $= v_i^-\sigma = v\sigma$.

(c) If $v \neq v_i, v \neq v_i^-$, for all $i = 1, \ldots, n$, then $v\delta_{P'}\sigma = v\sigma$. □

We now prove the soundness of the algorithm. If $P'$ is solved then by Lemma 8.3, $\delta_{P'}$ is an idempotent *mgu* of $P'$. Therefore, it is sufficient to show that if $P \Rightarrow^* P'$ then $P$ and $P'$ are equivalent (i.e., they have the same unifiers). First observe that the equivalence is stepwise preserved by the relation $\Rightarrow$. In fact, steps (1)-(4) (6) and (7) clearly do not affect the set of unifiers. Then assume

$$P = \{\langle t_1, u_1\rangle, \ldots, \langle t_n, u_n\rangle\} \;\Rightarrow\; P' = \{\langle t_1', u_1'\rangle, \ldots, \langle t_n', u_n'\rangle\}$$

via step (5). Let $\langle t_i, u_i\rangle$ be the selected pair in $P$. Then, $t_i = x \in Var$ and $t_j' = t_j\{x/u_i\}$, $u_j' = u_j\{x/u_i\}$ for $j = 1, \ldots, i-1, i+1, \ldots, n$. If $\vartheta$ is a unifier of $P$, then $x\vartheta = u_i\vartheta$ and $x^-\vartheta = u_i^-\vartheta$. Therefore $\{x/u\}\vartheta = \vartheta$. Thus we have $t_j'\vartheta = (t_j\{x/u_i\})\vartheta = t_j(\{x/u_i\}\vartheta) = t_j\vartheta =$ (since $\vartheta$ is a unifier of $P$) $= u_j\vartheta = (u_j\{x/u_i\})\vartheta = u_j(\{x/u_i\}\vartheta) = u_j'\vartheta$, i.e., $\vartheta$ is a solution of $P'$. Analogously, if $\vartheta$ is a solution of $P'$, then $\vartheta$ is a solution of $P$.

(4) (*completeness*) In order to prove the completeness of the algorithm we need the following lemmas.

**Lemma 8.4.** *Let $M$ be a set of sets of terms. If $\vartheta$ is an idempotent most general unifier of $M$, then $\vartheta$ is* relevant (*see* [1]). *Namely, $\vartheta$ involves only the variables occurring in $M$, and their annotated versions, i.e.,*

$$\mathcal{D}(\vartheta) \cup \mathcal{D}(\vartheta) \subseteq \mathcal{V}(M) \cup \mathcal{V}(M)^-.$$

**Proof.** By Proposition 8.2(1), (2) and (3), if $M$ is unifiable, then there exists a set of pairs $P$ such that:

- $symm(M_{pairs}) \Rightarrow^* P$,
- $P$ is in solved form,
- $\delta_P \in mgu(M)$.

By definition, it follows immediately that $\delta_P$ is relevant. Since $\vartheta$ is a most general unifier of $M$, there exists $\mu$ such that $\vartheta\mu = \delta_P$. Then, we have: $\vartheta\delta_P = \vartheta\vartheta\mu =$ (since $\vartheta$ is idempotent) $= \vartheta\mu = \delta_P$. It is easy to see that

$$\mathcal{D}(\vartheta)\backslash\mathcal{C}(\delta_P) \subseteq \mathcal{D}(\vartheta\delta_P) = \mathcal{D}(\delta_P) \quad \text{and} \quad \mathcal{C}(\vartheta)\backslash\mathcal{D}(\delta_P) \subseteq \mathcal{C}(\vartheta\delta_P) = \mathcal{C}(\delta_P).$$

hold. Moreover, since $\delta_P$ is relevant, we have

$$\mathcal{D}(\delta_P) \subseteq \mathcal{V}(M) \cup \mathcal{V}(M)^- \quad \text{and} \quad \mathcal{C}(\delta_P) \subseteq \mathcal{V}(M) \cup \mathcal{V}(M)^-.$$

Therefore

$$\mathcal{D}(\vartheta)) \subseteq \mathcal{V}(M) \cup \mathcal{V}(M)^- \quad \text{and} \quad \mathcal{C}(\vartheta) \subseteq \mathcal{V}(M) \cup \mathcal{V}(M)^-,$$

i.e., $\vartheta$ is relevant. □

**Lemma 8.5.** *If $\vartheta \sim \vartheta'$ (i.e. $\vartheta \leqslant \vartheta'$ and $\vartheta' \leqslant \vartheta$), then there exists a renaming $\rho$ such that $\vartheta' = \vartheta\rho$ and $\vartheta = \vartheta'\rho^{-1}$.*

**Proof.** It is an immediate extension of a lemma stated in [15]. See also [10] for an easy proof. □

**Lemma 8.6.** *If $\vartheta \in mgu(M)$, then*

$$mgu(M) = \{\vartheta' \mid \vartheta' \text{ is idempotent and } \exists\rho \text{ renaming: } \vartheta' = \vartheta\rho\}.$$

**Proof.** If $\vartheta, \vartheta' \in mgu(M)$, then $\vartheta \leqslant \vartheta'$ and $\vartheta' \leqslant \vartheta$, i.e., $\vartheta \sim \vartheta'$. By Lemma 8.5, we have $\vartheta' = \vartheta\rho$ for an appropriate renaming $\rho$. On the other side, if $\vartheta' = \vartheta\rho$, and $\vartheta \in mgu(M)$, then $\vartheta'$ is a unifier of $M$. Moreover, for any other $\sigma$ that unifies $M$, since $\vartheta \leqslant \sigma$, we have $\exists\rho$: $\vartheta\tau = \sigma$. Thus $\vartheta'\rho^{-1}\tau = \vartheta\tau = \sigma$, i.e., $\vartheta' \leqslant \sigma$. □

We now prove the completeness of the algorithm. By Lemma 8.6, if $\vartheta, \vartheta' \in mgu(M)$, then $\vartheta' = \vartheta\rho$ for an appropriate $\rho$. By Lemma 8.4, $\rho$ does not introduce new variables. Then, we can decompose $\vartheta, \vartheta'$ into two parts:

$$\vartheta = \vartheta_1 \cup \vartheta_2, \qquad \vartheta' = \vartheta_1' \cup \vartheta_2',$$

such that

$$\vartheta_1 = \{v_1/w_1, \ldots, v_n/w_n\}, \qquad \vartheta_1' = \{w_1/v_1, \ldots, w_n/v_n\},$$
$$\rho = \vartheta_1 \cup \vartheta_1' \quad \text{and} \quad \vartheta_2' = \vartheta_2\vartheta_1'.$$

Now observe that $M_{pairs}$ is symmetric, i.e., $\langle t, u\rangle \in M_{pairs}$ iff $\langle u, t\rangle \in M_{pairs}$. Moreover it is easy to see that if $symm(M_{pairs}) \Rightarrow^* P$ and $\langle t_1, u_1\rangle, \ldots, \langle t_n, u_n\rangle \in P$, then $symm(M_{pairs}) \Rightarrow^* P' = P \cup \{\langle u_1, t_1\rangle, \ldots, \langle u_n, t_n\rangle\}$. Now let

$$P_1 = \{\langle t, u\rangle \mid t/u \in \vartheta_1\} = \{\langle v_1, w_1\rangle, \ldots, \langle v_n, w_n\rangle\},$$
$$P_2 = \{\langle t, u\rangle \mid t/u \in \vartheta_2'\},$$
$$P_1' = \{\langle t, u\rangle \mid t/u \in \vartheta_1'\} = \{\langle w_1, v_1\rangle, \ldots, \langle w_n, v_n\rangle\},$$
$$P_2' = \{\langle t, u\rangle \mid t/u \in \vartheta_2'\},$$

and assume

$$symm(M_{pairs}) \Rightarrow^* P_1 \cup P_2 = \{\langle v_1, w_1\rangle, \ldots, \langle v_n, w_n\rangle\} \cup P_2.$$

Then

$$\begin{aligned} symm(M_{pairs}) \Rightarrow^* & \{\langle v_1, w_1\rangle, \ldots, \langle v_n, w_n\rangle\} \\ & \cup \{\langle w_1, v_1\rangle, \ldots, \langle w_n, v_n\rangle\} \cup P_2, \end{aligned}$$

and since $\{\langle v_1, w_1\rangle, \ldots, \langle v_n, w_n\rangle\}\{w_1/v_1, \ldots, w_n/v_n\} = \{\langle v_1, v_1\rangle, \ldots, \langle v_n, v_n\rangle\}$, which is eliminated by step (3), we have

$$\begin{aligned} symm(M_{pairs}) \Rightarrow^* & \{\langle w_1, v_1\rangle, \ldots, \langle w_n, v_n\rangle\} \cup P_2\{w_1/v_1, \ldots, w_n/v_n\} \\ & = \{\langle w_1, v_1\rangle, \ldots, \langle w_n, v_n\rangle\} \cup P_2'. \end{aligned}$$

(5) (*soundness and completeness of failure*) We want to show that the algorithm fails iff the initial set $P$ is not unifiable.

*if part*: By part (1) and (2) of this proposition, either $P \Rightarrow^* P'$, where $P'$ is in solved form, or $P \Rightarrow^* fail$. By part (3), the first case implies that $P$ is unifiable, therefore $P \not\Rightarrow fail$.

*only-if part*: Assume $P \Rightarrow^* fail$. Let $P'$ be the set of pairs such that $P \Rightarrow^* P'$ and $P' \Rightarrow fail$. Then, one of steps (2), (5) (first case), or (7) (first case) applies to $P'$, i.e.,

- $\langle f(t_1, \ldots, t_n), g(u_1, \ldots, u_n)\rangle$, where $f \neq g$, or
- $\langle x, t\rangle$, where $x \in Var$, $x \neq t$, $x^- \neq t$ and $(x \in \mathcal{V}(t)$ or $x^- \in \mathcal{V}(t))$, or
- $\langle x^-, t\rangle$, where $x^- \in Var^-$, $x^- \neq t$ and $x^- \in \mathcal{V}(t)$.

In all cases, $P'$ is clearly not unifiable. Since $\Rightarrow$ preserves the equivalence (see the proof of part (3) of this proposition), $P'$ is equivalent to $P$. Therefore, $P$ is also not unifiable. □

## References

[1] K.R. Apt, Introduction to logic programming (revised and extended version, Technical Report CS-R8826, Centre for Mathematics and Computer Science, Amsterdam, 1988; also in: J. Van Leeuwen, ed., *Handbook of Theoretical Computer Science Vol. B* (Elsevier, Amsterdam, 1990) 493-574.

[2] P. America and J.J.M.M. Rutten, Solving reflexive domain equations in a category of complete metric spaces, *J. Comput. System Sci.* **39**(3) (1989) 343-375.

[3] L. Beckman, Towards a formal semantics for concurrent logic programming languages, in: E. Shapiro, ed. *Proc. 3rd Internat. Conf on Logic Programming*, **225** (Springer, Berlin, 1986) 335-349.

[4] K.L. Clark and S. Gregory, Notes on the implementation of parlog, *J. Logic Programming* **2**(1) (1985) 17-42.

[5] K.L. Clark and S. Gregory, Parlog: parallel programming in logic, *ACM TOPLAS* **8**(1) (1986) 1-49.

[6] J.W. de Bakker and J.N. Kok, Uniform abstraction, atomicity and contractions in the comparative semantics of concurrent prolog, in: *Proc. 5th Generation Computer Systems*, Extended Abstract. (Ohmsha Ltd, Tokyo, Japan, 1988) 347-355; full version in CWI report CS-8834; *Theoret. Comput. Sci.*, to appear.

[7] F.S. de Boer, J.N. Kok, C. Palamidessi and J.J.M.M. Rutten, Control flow versus logic: a denotational and a declarative model for guarded horn clauses, in: A. Kreczmar and G. Mirkowska, eds., *Proc. Mathematical Foundations of Computer Science* (*MFCS 89*), **379** (Springer, Berlin, 1989) 165-177.

[8] F.S. de Boer, J.N. Kok, C. Palamidessi and J.J.M.M. Rutten, From failure to success: comparing a denotational and a declarative semantics for horn clause logic, in: *Proc. Internat. BCS-FACS Workshop on Semantics for Concurrency*, Leicester (1990), to appear in *Theoret. Comput. Sci.*

[9] J.W. de Bakker and J.I. Zucker, Processes and the denotational semantics of concurrency, *Inform. and Control* **54** (1982) 70-120.

[10] E. Eder, Properties of substitutions and unifications, *J. Symbolic Comput.* **1** (1985) 31-46.

[11] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi, A new declarative semantics for logic languages, in: K.A. Bowen and R.A. Kowalski, eds., *Proc. 5th Conf. and Symp. on Logic Programming*, Seattle (MIT Press, Cambridge, MA, 1988) 993-1005.

[12] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi, Declarative modeling of the operational behaviour of logic languages, *Theoret. Comput. Sci.* **69**(3) (1989) 289-318.

[13] S. Gregory, *Parallel Logic Programming in PARLOG*, International Series in Logic Programming (Addison-Wesley, Reading, MA, 1987).

[14] M. Hennesy and G.D. Plotkin, Full abstraction for a simple parallel programming language, in: J. Becvar, ed., *Proc. 8th Internat. Symp. on Mathematical Foundations on Computer Science*, Lecture Notes in Computer Science **74** (Springer, Berlin, 1979) 108-120.

[15] G. Huet, Resolution d'Equations dans des Langages d'Order 1, 2, ..., $\omega$, PhD thesis, Univ. Paris VII, 1976.

[16] G. Levi, Models, unfolding rules and fixed point semantics, in: K.A. Bowen and R.A. Kowalski, ed., *Proc. 5th Conf. and Symp. on Logic Programming,* Seattle (MIT Press, Cambridge, MA, 1988) 1649-1665.

[17] J.-L. Lassez, M.J. Maher and K. Marriot, Unification revisited, in: J. Minker, ed. *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, 1988).

[18] G. Levi and C. Palamidessi, The declarative semantics of logical read-only variables, in: *Proc. IEEE Symp. on Logic Programming,* Boston (IEEE Computer Society Press, 1985) 128-137.

[19] G. Levi and C. Palamidessi, An approach to the declarative semantics of synchronization in logic languages, in: J.-L. Lassez, ed., *Proc. 4th Internat. Conf. on Logic Programming,* Melbourne (MIT Press, Cambridge, MA, 1987) 877-893.

[20] M. Murakami, A declarative semantics of parallel logic programs with perpetual processes, in: *Proc. 5th Generation Computer Systems,* (Ohmsha Ltd, Tokyo, Japan, 1988) 374-381.

[21] C. Palamidessi, A fixpoint semantics for guarded horn clauses, Technical Report CS-R8833, Centre for Mathematics and Computer Science, Amsterdam, 1988.

[22] V.A. Saraswat, Partial correctness semantics for cp($\downarrow, |, \&$), in: *Proc. Conf. on Foundations of Software Computing and Theoretical Computer Science,* **206** (Springer, Berlin, 1985) 347-368.

[13] V.A. Saraswat, The concurrent logic programming language cp: definition and operational semantics, In: *Conf. Record of the 14th Ann. ACM Symp. on Principles of Programming Languages* (ACM, New York, 1987) 49-63.

[24] V.A. Saraswat, GHC: operational semantics, problems and relationship with cp($\downarrow, |$), in: *IEEE Internat. Symp. on logic programming,* San Francisco (IEEE, New York, 1987) 347-358.

[25] E.Y. Shapiro, A subset of concurrent prolog and its interpreter, Technical Report TR-003, ICOT, 1983.

[26] E.Y. Shapiro, *Concurrent Prolog: Collected Papers,* Vols. 1, 2 (MIT Press, Cambridge, MA, 1988).

[27] A. Takeuchi and K. Furukawa, Parallel logic programming languages, in: E. Shapiro, ed., *Proc. 3rd Internat. Conf. on Logic Programming,* **225** (Springer, Berlin, 1986).

[28] K. Ueda, Guarded horn clauses, Technical Report TR-103, ICOT, Tokyo, 1985; revised in 1986; a revised version is in: E. Wada, ed., *Proc. Logic Programming '85,* Lecture Notes in Computer Science **221** (Springer, Berlin, 1986) 335-349; also in: E.Y. Shapiro, ed., *Concurrent Prolog: Collected Papers* (MIT Press, Cambridge, MA, 1988) Chap. 4.

[29] K. Ueda, Guarded horn clauses, a parallel logic programming language with the concept of a guard, in: M. Nivat and K. Fuchi, eds., *Programming of Future Generation Computers* (North-Holland, Amsterdam, 1988) 441-456.

[30] M.H. van Emden and R.A. Kowalski, The semantics of predicate logic as a programming language, *J. ACM* **23**(4) (1976) 733-742.