Centrum voor Wiskunde en Informatica

Centre for Mathematics and Computer Science

F.S. de Boer, J.N. Kok, C. Palamidessi, J.J.M.M. Rutten

Semantic models for a version of Parlog

Computer Science/Department of Software Technology Report CS-R8951 January

1989

Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

F.S. de Boer, J.N. Kok, C. Palamidessi, J.J.M.M. Rutten

Semantic models for a version of Parlog

Computer Science/Department of Software Technology   Report CS-R8951   January

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Semantic Models for a Version of PARLOG *

Frank S. de Boer[1]
Joost N. Kok[2]
Catuscia Palamidessi[3]
Jan J.M.M. Rutten[1]

[1]Centre for Mathematics and Computer Science,
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

[2]Department of Computer Science, University of Utrecht,
P.O. Box 80089, 3508 TB Utrecht, The Netherlands

[3]Dipartimento di Informatica, Università di Pisa,
Corso Italia 40, 56100 Pisa, Italy

Abstract

This paper gives four semantics for PARLOG: 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 of a PARLOG program, that is, the set of computed answer substitutions corresponding to all successfully terminating computations. The other operational and the denotational semantics model also deadlock and infinite computations. For the declarative and the denotational semantics we extend standard notions like unification in order to cope with the synchronization mechanism of PARLOG. The basic mathematical structure for the declarative semantics is the set of finite streams of substitutions. In the denotational semantics we use tree-like structures that are labelled with streams of substitutions. We look at the relations between the different models: First we relate the two operational semantics and next we show the relation of the declarative and denotational semantics with their operational counterparts.

Key words and phrases: operational semantics, denotational semantics, declarative semantics, parallelism, concurrent logic languages, correctness, complete metric spaces.
1985 Mathematics Subject Classification: 68Q55, 68Q10.
1987 Computing Reviews Categories: D.1.3, D.3.1, F.1.2, F.3.2.

1 Introduction.

The language PARLOG [CG85, CG86, Gre87], as well as most of the concurrent logic languages, is based on the Horn Clause Logic (HCL) plus some mechanisms for expressing concurrency. One of the main drawbacks of this approach is that these new mechanisms heavily affect the clean declarative understanding of HCL. Indeed, although many operational semantics have been investigated ([Sar85, Sar87a, Sar87b, Be86, GCLS88, BK88]), a satisfactory declarative one is still to be defined. PARLOG belongs to a class of concurrent logic languages whose main features are:

---

*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.

Report CS-R8951
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

- the *input-constraints*, on which the mechanism of synchronization between AND-processes is based, and
- the presence of *commit*, that realizes the *don't know nondeterminism*, controlled by guards.

Other languages in this class are Guarded Horn Clauses [U85, U88], Concurrent Prolog [Sh83, Sh88], and their flat versions. These mechanisms affect the semantics of the pure underlying language in several ways [TF86]:

- 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.

In this paper we address the problem of characterizing these new sets, for PARLOG, 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). We deal with a version of PARLOG. For example, we do not consider unification related primitives, OR-parallel aspects and local deadlock in deep guards.

First, we describe these sets by a formal operational semantics based on a transition relation (in the style of [HP79], see also [Sar87a, GCLS88, BK88] 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 fixpoint of an *immediate consequence* operator on interpretations. (The full model-theoretic semantics is still under investigation.) Our approach can be considered an evolution of the one developed in [LP85] and [LP87]. The basic 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 [FLPM88b] and [FLPM88a]). 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 always can. Now it is the case that the approach presented in [LP85] and [LP87] is not able to distinguish between the two situations. Indeed, $r(a^+, b^+)$ (r producing a and b) happens there to be *true* in the models (and in the least fixpoint 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 [L88].

Our solution to this problem consists in enriching the interpretations with streams of substitutions. Due to the presence of guards, whose evaluation has to be interpreted as an atomic action (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. An other 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 ([Ede85, LMM88]) 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, that allows us to model the combination of the substitutions computed by and-parallel processes.

Other compositional models for the success set are presented in [Sar85] and [M88]. 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 $x/f(x_1, ..., x_n)$. 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 characterizing the finite failures and the infinite computations in a compositional way. It turns out that the streams of the declarative semantics offer again too little structure, but also the sequences introduced in the declarative semantics are not powerfull enough. Indeed, 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.

To justify this, let's illustrate how the absence of nondeterminism informations (branching informations) causes our operational semantics to be *not* 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 $D_1 = \{p(?), q(?), r(?), s(\hat{})\}$, 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 $s(x)$, 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 RwrongS clause for $p(y)$), whereas in P_2 it cannot.

A possible way to provide these branching informations is to use tree-like structures. An open problem still with respect to compositionality is the minimal amount of information needed to obtain a compositional semantics. In [GCLS88] is described a fully abstract semantics for Flat-Concurrent Prolog. Their approach is based on suspension sets which are a more abstract structure than the tree-like one. But to obtain full abstraction they also incorporate in their semantic description an operation abstracting from unifications generating only bindings to local variables. It is however known in the world of concurrent imperative languages that to obtain a fully abstract semantics based on suspension sets for a language embodying a kind of abstraction operator is quite a hard problem. It seems that this problem in some form should also occur in the semantic description of Concurrent Prolog based on suspension sets. Moreover, it is also not clear how it can be extended to the general case of non-flat guards. The same applies to the declarative approach taken in [FL88] to characterize the finite failures.

In our approach we code the branching informations by using trees labelled with streams of substitutions. We see them as elements of complete metric spaces, satisfying so-called reflexive domain equations ([BZ82], [AR89]). We use a *denotational* style: for every operator in the language we define a semantic operator that can be seen as a function on these spaces. The meaning of a goal can then be given by a semantic operator that results to be the *unique* fixpoint of a *contraction* on the functional metric spaces ([BZ82], [AR89]). The relation of this denotational semantics with respect to the operational one is obtained via an *abstraction operator*, that identifies some denotations. The correctness of the denotational semantics is then stated as the equality between the result of this abstraction and the operational semantics.

The definition of PARLOG has been changed with respect to the previous versions. We consider the one described in [Gre87].

2 The language PARLOG

To describe the syntax of the language PARLOG 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 $\Box$ we denote the goal $\leftarrow$ *true*. A clause is of the form $C = H \leftarrow \bar{G} | \bar{B}$, 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 backtraking 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 this 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 [LMM88]), with respect to V, of the clauses of W. We introduce the set of substitutions ($\vartheta, \gamma \in$) *Subst*. ϵ is the *empty substitution*. For V a finite set of variables, we use $\vartheta_{|V}$ to denote the restriction of ϑ 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* mgu's: Consider two atoms

$A = p(t_1, ..., t_n)$ and $A' = p(t'_1, ..., t'_n)$. Assume that the declaration-mode of p has the symbol ? (input-mode) on the arguments $i_1, ..., i_k$. 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

$$\begin{array}{lll}
1. & \text{If} & \exists H \leftarrow \bar{G} | \bar{B} \in W_{\mathcal{V}(A)}, \exists mgu_i(A\vartheta, H) \\
 & & [<\leftarrow \bar{G}, mgu_i(A\vartheta, H) > \overset{*}{\rightarrow} < \Box, \vartheta' >, \ and \ \vartheta'|_{invar(A\vartheta)} = \epsilon], \\
 & \text{then} & <\leftarrow A, \vartheta > \rightarrow <\leftarrow outunif(A\vartheta, H\vartheta'), \bar{B}, \vartheta\vartheta' > \\
 & & \\
2. & \text{If} & \exists mgu_o(A\vartheta, H\vartheta'), \\
 & \text{then} & <\leftarrow outunif(A\vartheta, H\vartheta'), \bar{B}, \vartheta' > \rightarrow <\leftarrow \bar{B}, \vartheta' mgu_o(A\vartheta, H\vartheta') > . \\
 & & \\
3. & \text{If} & <\leftarrow \bar{A}, \vartheta > \rightarrow <\leftarrow \bar{A}', \vartheta' > \mid < \Box, \vartheta' > \\
 & \text{then} & <\leftarrow \bar{A}, \bar{B}, \vartheta > \rightarrow <\leftarrow \bar{A}', \bar{B}, \vartheta' > \mid <\leftarrow \bar{B}, \vartheta' > \\
 & & <\leftarrow \bar{B}, \bar{A}, \vartheta > \rightarrow <\leftarrow \bar{B}, \bar{A}', \vartheta' > \mid <\leftarrow \bar{B}, \vartheta' >
\end{array}$$

□

In these transitions, ϑ 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 ϑ' must not instantiate any input variables of $A\vartheta$. The transition 2. represents the first action performed after the commitment, namely the output-unification. 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

$$\begin{array}{ll}
\mathcal{O}_1 : Goal \rightarrow M_1, & \text{with } M_1 = \mathcal{P}(Subst) \\
\mathcal{O}_2 : Goal \rightarrow M_2, & \text{with } M_2 = \mathcal{P}(Subst^{\infty}_{\delta}).
\end{array}$$

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

We put $\mathcal{O}_i\llbracket\leftarrow true\rrbracket = \{\epsilon\}$, and

$$\begin{array}{rcl}
\mathcal{O}_1\llbracket\leftarrow \bar{A}\rrbracket & = & \{\vartheta_{|\mathcal{V}(\bar{A})} \mid <\leftarrow \bar{A}, \epsilon > \overset{*}{\rightarrow} < \Box, \vartheta >\}; \\
 & & \\
\mathcal{O}_2\llbracket\leftarrow \bar{A}\rrbracket & = & \{(\vartheta_1. \cdots \vartheta_n)_{|\mathcal{V}(\bar{A})} \in Subst^{+} \mid \\
 & & \qquad <\leftarrow \bar{A}, \epsilon > \rightarrow <\leftarrow \bar{A}_1, \vartheta_1 > \rightarrow \cdots \rightarrow < \Box, \vartheta_n >\} \\
 & \cup & \{(\vartheta_1. \cdots \vartheta_n)_{|\mathcal{V}(\bar{A})}.\delta \in Subst^{*}.\{\delta\} \mid \\
 & & \qquad <\leftarrow \bar{A}, \epsilon > \rightarrow \cdots \rightarrow < \bar{A}_n, \vartheta_n > \not\rightarrow \wedge \leftarrow \bar{A}_n \neq \Box\} \\
 & \cup & \{(\vartheta_1. \cdots)_{|\mathcal{V}(\bar{A})} \in Subst^{\omega} \mid <\leftarrow \bar{A}, \epsilon > \rightarrow <\leftarrow \bar{A}_1, \vartheta_1 > \rightarrow \cdots\}.
\end{array}$$

□

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

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

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

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.

In this section we define the declarative (fixpoint) semantics of PARLOG. We make use of an extended notion of Herbrand base and interpretations, enriched with variables (that are used for modeling the notion of *computed substitution*, [LP87],[FLPM88b],[FLPM88a]), and *annotations* (that are used for modeling the synchronization mechanism of concurrent logic languages, see [LP85] and [LP87] for similar approaches). We extend the standard notions of the unification theory ([Ede85], [LMM88]) in a formal framework. Moreover, we introduce the notion of *parallel composition*, that allows 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 to overcome the difficulties presented in [LP87] concerning deadlock. A similar construction has been made for defining the declarative semantics of Guarded Horn Clauses ([BKPR88a], [Pal88]).

4.1 Annotated variables.

In order to model the synchronization mechanism of PARLOG 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 ϑ is a mapping $\vartheta : Var \cup Var^- \longrightarrow Term$, such that $\vartheta(v) \neq v$ for finitely many v only. ϑ 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 ϑ to a term (or atom, or formula) t:

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

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 ϑ 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 ϑ to a different term). An annotated variable mapped to a different term represents a violation of the associated input-mode constraint. 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 ϑ will indicate its equivalence class.

Example 4.1 Consider the atom $A = p(f(x, y), x, y)$. We annotate the variables in A so to get $A^- = p(f(x^-, y^-), x^-, y^-)$. 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, ϑ_1 and ϑ_2 is extended as follows

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

The composition is associative and the empty substitution ϵ 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:

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

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

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

Given a set of sets of terms M, we define ϑ to be a unifier for M iff

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

The ordering on substitutions is the standard one, namely: $\vartheta_1 \leq \vartheta_2$ *iff* $\exists \vartheta_3\ [\vartheta_1\vartheta_3 = \vartheta_2]$ (ϑ_1 *is more general than* ϑ_2). The set of idempotent mgu's (most general unifiers) of a set of sets of terms M is denoted by $mgu(M)$.

We give an extended version of the unification algorithm, based on the one presented in [Apt87], that works on finite sets of pairs. Given a finite set of finite sets of terms M, consider the (finite) set of pairs

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

We define the unifiers of a set $\{< t_1, u_1 >, \ldots, < t_n, u_n >\}$ 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

$$\{< v_1, t_1 >, \ldots, < v_n, t_n >\}$$

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

  We will write $P \Rightarrow \mathit{fail}$ 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 $P_{sol} = \{P' \mid symm(P) \Rightarrow^* P', \text{ and } P' \text{ is solved}\}$, where

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

  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 4.3 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 the 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 $P_1 \Rightarrow P_2 \Rightarrow \dots P_n \Rightarrow \dots$. 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 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 4.4 Let P be a set of pairs. Let $P' \in P_{sol}$. Then $\delta_{P'} \in mgu(P')$.

   Proof Let $P' = \{< v_1, t_1 >, \dots, < v_n, t_n >\} \in P_{sol}$. 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 $i = 1, \dots, n$. 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 $v\delta_{P'}\delta_{P'} = t_i^-\gamma_{P'}\gamma_{P'}\gamma_{P'} =$ (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 $i = 1, \dots, n$, then $v\delta_{P'}\delta_{P'} = v\delta_{P'}(= v)$.

   (unifier) For each $i = 1, \dots n$, 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 σ 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 σ 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 σ is a unifier of P') $= t_i^-\sigma =$ (since σ 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 prove now the soundness of the algorithm. If P' is solved then by lemma 4.4, $\delta_{P'}$ 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

$$P = \{< t_1, u_1 >, \ldots, < t_n, u_n >\} \Rightarrow P' = \{< t'_1, u'_1 >, \ldots, < t'_n, u'_n >\}$$

via step *(5)*. Let $< t_i, u_i >$ 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 ϑ 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 ϑ 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., ϑ is a solution of P'. Analogously, if ϑ is a solution of P', then ϑ is a solution of P.

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

Lemma 4.5 Let M be a set of sets of terms. If ϑ is an idempotent most general unifier of M, then ϑ is *relevant* (see [Apt87]). Namely, ϑ 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 4.3*(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 δ_P is relevant. Since ϑ is a most general unifier of M, there exists μ such that $\vartheta\mu = \delta_P$. Then, we have: $\vartheta\delta_P = \vartheta\vartheta\mu =$ (since ϑ is idempotent) $= \vartheta\mu = \delta_P$. It is easy to see that

$$\mathcal{D}(\vartheta)\backslash\mathcal{C}(\delta_P) \subseteq \mathcal{D}(\vartheta\delta_P) = \mathcal{D}(\delta_P)$$

and

$$\mathcal{C}(\vartheta)\backslash\mathcal{D}(\delta_P) \subseteq \mathcal{C}(\vartheta\delta_P) = \mathcal{C}(\delta_P)$$

hold. Moreover, since δ_P is relevant, we have

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

and

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

Therefore

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

and

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

i.e., ϑ is relevant. □

Lemma 4.6 If $\vartheta \sim \vartheta'$ (i.e. $\vartheta \leq \vartheta'$ and $\vartheta' \leq \vartheta$), then there exists a renaming ρ such that $\vartheta' = \vartheta\rho$ and $\vartheta = \vartheta'\rho^{-1}$.

Proof It is an immediate extension of a lemma stated by Huet in ([Hu]). See also [Ede85] for an easy proof. □

Lemma 4.7 If $\vartheta \in mgu(M)$, then

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

Proof If $\vartheta, \vartheta' \in mgu(M)$, then $\vartheta \leq \vartheta'$ and $\vartheta' \leq \vartheta$, i.e., $\vartheta \sim \vartheta'$. By lemma 4.6, we have $\vartheta' = \vartheta\rho$ for an appropriate renaming ρ. On the other side, if $\vartheta' = \vartheta\rho$, and $\vartheta \in mgu(M)$, then ϑ' is a unifier of M. Moreover, for any other σ that unifies M, since $\vartheta \leq \sigma$, we have $\exists\tau : \vartheta\tau = \sigma$. Thus

$$\vartheta'\rho^{-1}\tau = \vartheta\tau = \sigma,$$

i.e., $\vartheta' \leq \sigma$. □

We prove now the completeness of the algorithm. By lemma 4.7, if $\vartheta, \vartheta' \in mgu(M)$, then $\vartheta' = \vartheta\rho$ for an appropriate ρ. By lemma 4.5, ρ does not introduce new variables. Then, we can decompose ϑ, ϑ' into two parts:

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

$$\vartheta' = \vartheta'_1 \cup \vartheta'_2,$$

such that:

$$\vartheta_1 = \{v_1/w_1, \ldots, v_n/w_n\},$$

$$\vartheta'_1 = \{w_1/v_1, \ldots, w_n/v_n\},$$

$$\rho = \vartheta_1 \cup \vartheta'_1,$$

and

$$\vartheta'_2 = \vartheta_2\vartheta'_1.$$

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

$$P_1 = \{< t, u > \mid t/u \in \vartheta_1\} = \{< v_1, w_1 >, \ldots, < v_n, w_n >\},$$

$$P_2 = \{< t, u > \mid t/u \in \vartheta'_2\},$$

$$P_1' = \{< t, u > \mid t/u \in \vartheta_1'\} = \{< w_1, v_1 >, \ldots, < w_n, v_n >\},$$

$$P_2' = \{< t, u > \mid t/u \in \vartheta_2'\},$$

and assume

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

Then

$$symm(M_{pairs}) \Rightarrow^* \{< v_1, w_1 >, \ldots, < v_n, w_n >\} \cup \{< w_1, v_1 >, \ldots, < w_n, v_n >\} \cup P_2,$$

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

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

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.,

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

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

□

4.2 Parallel composition on substitutions.

In this section we introduce the notion of *parallel composition* on substitutions and on 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 ϑ_1 and ϑ_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 ϑ_1 and ϑ_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 4.8 In the following, $\mathcal{S}(\vartheta)$ is the set of sets $\{\{x, t\} \mid x/t \in \vartheta\}$. Θ_1, Θ_2 are sets of substitutions.

1. $\vartheta_1 \hat{\circ} \vartheta_2 = mgu(\mathcal{S}(\vartheta_1) \cup \mathcal{S}(\vartheta_2))$.
2. $\Theta_1 \hat{\circ} \Theta_2 = \bigcup_{\vartheta_1 \in \Theta_1, \vartheta_2 \in \Theta_2} \vartheta_1 \hat{\circ} \vartheta_2$.

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

Example 4.9

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
   $mgu(p(x^-), p(f(a))) = \{\vartheta_1\}$, where $\vartheta_1 = \{x^-/f(a)\}$, and
   $mgu(q(x), q(f(a))) = \{\vartheta_2\}$, where $\vartheta_2 = \{x/f(a)\}$.
   Now observe that since $\vartheta_1 \in mgu(\mathcal{S}(\vartheta_1))$, $\vartheta_2 \in mgu(\mathcal{S}(\vartheta_2))$ and $\vartheta_1 \leq \vartheta_2$ we have $\vartheta_2 \in \vartheta_1 \hat{\circ} \vartheta_2$.

2. Consider now the same program and goal as before, but let the declaration mode be $\{p(?), q(?)\}$. We have $mgu(p(x^-), p(f(a))) = mgu(q(x^-), q(f(a))) = \{\vartheta_1\}$, and $\vartheta_1 \in \vartheta_1 \hat{\circ} \vartheta_1$, whilst $\vartheta_2 \notin \vartheta_1 \hat{\circ} \vartheta_1$.

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 ϑ_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 [FLPM88b] and [FLPM88a], the computed bindings in HCL can be declaratively modeled 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 [LP85] and [LP87], 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 a too weak 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 4.10 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 delimitate the *critical sections.* 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 $S.S' \overset{def}{=} \{s.s' \mid s \in S, s' \in S'\}$ and $[S]_V \overset{def}{=} \{[s]_V \mid s \in S\}$. If $s = \vartheta'.s'$, then $\vartheta \hat{\circ} s \overset{def}{=} (\vartheta \hat{\circ} \vartheta').s'$ and $\vartheta \hat{\circ} ([s]_V.s'') \overset{def}{=} [(\vartheta \hat{\circ} \vartheta').s']_V.s''$. For Θ a set of substitution we have $\Theta \hat{\circ} s \overset{def}{=} \bigcup_{\vartheta \in \Theta} \vartheta \hat{\circ} s$.

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. $\exists k : \forall s \in S : \#(s) = k$, then we define $\#(S) = k$.

Definition 4.11 (Interleaving operator).

1. $$\begin{array}{lcl} s_1 \parallel s_2 & = & (s_1 \mathbin{\|\!\!\lfloor} s_2) \cup (s_2 \mathbin{\|\!\!\lfloor} s_1) \\ (\vartheta.s_1) \mathbin{\|\!\!\lfloor} s_2 & = & \vartheta.(s_1 \parallel s_2) \\ ([s]_V.s_1) \mathbin{\|\!\!\lfloor} s_2 & = & [s]_V.(s_1 \parallel s_2) \end{array}$$
2. $$S_1 \parallel S_2 = \bigcup_{s_1 \in S_1, s_2 \in S_2} s_1 \parallel s_2.$$

□

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 4.12

1. $\mathcal{R}(\vartheta) = \begin{cases} \{\vartheta\} & \text{if } \vartheta_{|Var^-} = \epsilon \\ \emptyset & \text{otherwise} \end{cases}$
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. $\mathcal{R}(s_1.s_2) = \mathcal{R}(\mathcal{R}(s_1) \,\hat{\circ}\, s_2)$
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 ϑ (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 $\mathcal{R}(\vartheta_1.\vartheta_2.\ldots.\vartheta_n) = \mathcal{R}(\ldots\mathcal{R}(\mathcal{R}(\vartheta_1) \,\hat{\circ}\, \vartheta_2) \ldots \,\hat{\circ}\, \vartheta_n)$.

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 ([vEK76],[Apt87]).

The *Herbrand base with variables* associated to the program W, denoted by $\mathcal{B}$, the set 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 $< A, s >$, where A is an atom in $\mathcal{B}$ and s is a sequence of substitutions on *Var* and *Term*. $< A, s > \in I$ 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, that will be used in the least fixpoint construction, is mainly introduced for technical reasons.

Definition 4.13 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 $A_1, \ldots, A_k$ iff

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

□

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 $< \bar{A}, \bar{s} >$ stands for $< A_1, s_1 >, \ldots, < A_n, s_n >$, and $\| (\bar{s})$ stands for $s_1 \| \cdots \| s_n$.

Definition 4.14 The mapping $T : \mathcal{I} \rightarrow \mathcal{I}$, associated to W, is defined as follows:

$$\begin{aligned} T(I) = \quad \{< A, s >| \quad & \exists H \leftarrow \bar{G} | \bar{B} \in W_{\mathcal{V}(A)}, \\ & \exists \bar{s}', \bar{s}'' \text{ locally independent on } \bar{G}, \bar{B}, A, \\ & < \bar{G}, \bar{s}' >, < \bar{B}, \bar{s}'' > \in I : \\ & s \in [mgu_i(A^-, H).(\| (\bar{s}'))]_V . mgu_o(A, H).(\| (\bar{s}''))\} \end{aligned}$$

□

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_o and a sequence resulting from the body.

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

Proposition 4.15 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 $\mathcal{F}(W) = lfp(T)$.

4.5 Equivalence results.

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

$$\begin{aligned} & \mathcal{O}_1(\leftarrow \bar{A}) = \\ & \{\vartheta \mid \exists \bar{s} \text{ locally independent on } \bar{A} : < \bar{A}, \bar{s} > \in \mathcal{F}(W) \text{ and } \vartheta \in \mathcal{R}(\| (\bar{s}))_{|\mathcal{V}(\bar{A})}\} \end{aligned}$$

The following proposition can easily be proved for the unification algorithm given in definition 4.2.

Proposition 4.16 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 4.17 Let A, H be atoms. Let μ be an idempotent positive substitution with no variables in common with H. Then

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

Proof It is a particular case of proposition 4.16*(2)*. □

Lemma 4.18 Let μ be a positive substitution. Let $\vartheta_1, \ldots, \vartheta_n$ be idempotent substitutions. We have

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

then

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

Proof Immediate. □

Lemma 4.19 Let W be a Parlog program. Let $\bar{A}$ be a sequence of goals, and let ρ be a positive renaming such that $\bar{A}\rho$ is a variant of $\bar{A}$. Then

$$\exists\bar{s}\ locally\ independent\ on\ \bar{A} :< \bar{A}, \bar{s} > \in T_W^k(\emptyset)$$

$$\Leftrightarrow$$

$$\exists\bar{s}'\ locally\ independent\ on\ \bar{A}\rho :< \bar{A}\rho, \bar{s}' > \in T_W^k(\emptyset)$$

and

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

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 $< A_i, s_i > \in T_W^k(\emptyset)$. 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 fulfil the requirement of local independence). Then we have $< A_1\rho, s_1' >, \ldots, < A_n\rho, s_n' > \in T_W^k(\emptyset)$. Moreover by lemmata 4.17 and 4.18 we have, for each $s \in (s_1 \parallel \cdots \parallel s_n)$,

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

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

Lemma 4.20 Let W be a Parlog program. Let $\bar{A}$ be a sequence of atoms, and let μ be a positive substitution. Then

$\exists \bar{s}$ *locally independent on* $\bar{A} : < \bar{A}, \bar{s} > \in T^k_W(\emptyset)$

$\Leftrightarrow$

$\exists \bar{s}'$ *locally independent on* $\bar{A}\mu : < \bar{A}\mu, \bar{s}' > \in T^k_W(\emptyset)$

and

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

and

$\#(Int(\bar{s})) = \#(Int(\bar{s}')).$

Proof

($\Rightarrow$) Let μ be a positive substitution, and let ρ 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 4.19, 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. $< \bar{A}\mu, \bar{s}' > \in T^k_W(\emptyset)$
3. $(\mu\mathcal{R}(Int(\bar{s}')))_{|\mathcal{V}(A) \cup \mathcal{D}(\mu)} =$
   (by lemmata 4.17 and 4.18)
   $(\mathcal{R}(Int(\mu\bar{s}')))_{|\mathcal{V}(A) \cup \mathcal{D}(\mu)} =$
   (since only the ϑ'_i*'s* $\in mgu_i(A_i\mu^-, H_i)$*have variables in common with* μ*)*
   $(\mathcal{R}(\cdots \| (\mu \; mgu_i(A_i\mu^-, H_i)) \| mgu_o(A_i\mu^-, H_i) \| \cdots \| Int(\bar{s}''')))_{|\mathcal{V}(A) \cup \mathcal{D}(\mu)} =$
   $(\mathcal{R}(\cdots \| ((\rho\rho^{-1}\mu) \hat{\circ} \; mgu_i(A_i\rho^-, H_i)) \| mgu_o(A_i\rho^-, H_i) \cdots \| Int(\bar{s}''')))_{|\mathcal{V}(A) \cup \mathcal{D}(\mu)} =$
   (since the s'''_i*'s have no variables in common with* ρ*)*
   $(\mathcal{R}(\cdots \| ((\rho\rho^{-1}\mu) \hat{\circ} \; mgu_i(A_i\rho^-, H_i)) \| mgu_o(A_i\rho^-, H_i) \| \cdots \| Int(\rho\bar{s}''')))_{|\mathcal{V}(A) \cup \mathcal{D}(\mu)} =$
   (by lemmata 4.17 and 4.18)
   $(\rho(\mathcal{R}(\cdots \| ((\rho^{-1}\mu) \hat{\circ} \; mgu_i(A_i\rho^-, H_i)) \| mgu_o(A_i\rho^-, H_i) \| \cdots \| Int(\bar{s}'''))))_{|\mathcal{V}(A) \cup \mathcal{D}(\mu)} =$
   (by lemma 4.19)
   $(\mathcal{R}(\cdots \| (\rho.(\rho^{-1}\mu).mgu_i(A_i\rho^-, H_i)) \| mgu_o(A_i\rho^-, H_i) \| \cdots \| Int(\bar{s}'')))_{|\mathcal{V}(A) \cup \mathcal{D}(\mu)} =$
   (since $\rho_{-1}\mu = \rho_{-1} \hat{\circ} \; \mu$*)*
   $(\mathcal{R}(\cdots \| (\rho.\rho^{-1}.\mu.mgu_i(A_i\rho^-, H_i)) \| mgu_o(A_i\rho^-, H_i) \| \cdots \| Int(\bar{s}'')))_{|\mathcal{V}(A) \cup \mathcal{D}(\mu)} =$
   $(\mathcal{R}(\cdots \| (\mu.mgu_i(A_i\rho^-, H_i)) \| mgu_o(A_i\rho^-, H_i) \| \cdots \| Int(\bar{s}'')))_{|\mathcal{V}(A) \cup \mathcal{D}(\mu)} =$
   $(\mathcal{R}(\mu.(Int(\bar{s}'))))_{|\mathcal{V}(A) \cup \mathcal{D}(\mu)}.$

($\Leftarrow$) Analogously.

□

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

Definition 4.21 (Transition relation, k steps)

1. If $\exists H \leftarrow \bar{G}|\bar{B} \in W_{\mathcal{V}(A)}, \exists mgu_i(A\vartheta, H)$
   $[<\leftarrow \bar{G}, mgu_i(A\vartheta, H)> \rightarrow^k <\Box, \vartheta'>$, *and* $\vartheta'|_{invar(A\vartheta)} = \epsilon]$,
   then $<\leftarrow A, \vartheta> \rightarrow^{k+1} <\leftarrow outunif(A\vartheta, H\vartheta'), \bar{B}, \vartheta\vartheta'>$
2. If $\exists mgu_o(A\vartheta, H\vartheta')$,
   then $<\leftarrow outunif(A\vartheta, H\vartheta'), \bar{B}, \vartheta'> \rightarrow_1 <\leftarrow \bar{B}, \vartheta' mgu_o(A\vartheta, H\vartheta')>$.
3. If $<\leftarrow \bar{A}, \vartheta> \rightarrow^k <\leftarrow \bar{A}', \vartheta'> \mid <\Box, \vartheta'>$
   then $<\leftarrow \bar{A}, \bar{B}, \vartheta> \rightarrow^k <\leftarrow \bar{A}', \bar{B}, \vartheta'> \mid <\leftarrow \bar{B}, \vartheta'>$
   and $<\leftarrow \bar{B}, \bar{A}, \vartheta> \rightarrow^k <\leftarrow \bar{B}, \bar{A}', \vartheta'> \mid <\leftarrow \bar{B}, \vartheta'>$
4. If $<\leftarrow \bar{A}, \vartheta> \rightarrow^{k_1} <\leftarrow \bar{A}', \vartheta'>$
   and $<\leftarrow \bar{A}', \vartheta'> \rightarrow^{k_2} <\leftarrow \bar{A}'', \vartheta''>$
   then $<\leftarrow \bar{A}, \vartheta> \rightarrow^{k_1+k_2} <\leftarrow \bar{A}'', \vartheta''>$

□

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

$$<\leftarrow \bar{A}, \vartheta> \rightarrow^+ <\leftarrow \bar{A}', \vartheta'>$$

if and only if

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

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

Theorem 4.22 (Soundness) Let W be a Parlog program, and let $\bar{A}$ be a sequence of atoms. Then

$$\mathcal{O}_1(\leftarrow \bar{A})$$

$$\subseteq$$

$$\{\vartheta \mid \exists \bar{s} \text{ locally independent on } \bar{A} : <\bar{A}, \bar{s}> \in \mathcal{F}(W) \textit{ and } \vartheta \in \mathcal{R}(\| (\bar{s}))_{|\mathcal{V}(\bar{A})}\}$$

Proof By induction on the number of steps k.

($k = 1$) Assume $<\leftarrow \bar{A}, \epsilon> \rightarrow^1 <\Box, \vartheta>$. 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 $\vartheta_{|\mathcal{V}(A)} = \epsilon$. Then $<A, \sigma> \in T^1_W(\emptyset)$ holds, for each $\sigma \in mgu_i(A^-, H)$. Since $\vartheta_{|\mathcal{V}(A)} = \epsilon$, there exists $\sigma \in mgu_i(A^-, H)$ such that σ does not map variables in A, i.e., σ is positive. Therefore $\mathcal{R}(\sigma)_{|\mathcal{V}(A)} = \{\epsilon\}$.

($k > 1$) Assume $<\leftarrow \bar{A}, \epsilon> \rightarrow^{k_1} <\bar{A}', \sigma\mu> \rightarrow^{k_2} <\Box, \sigma\mu\vartheta'>$ 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 $H \leftarrow \bar{G}|\bar{B}$ in $W_{\mathcal{V}(\bar{A})}$ such that:

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

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

$$< \bar{G}\sigma, \bar{s}' > \in lfp(T_W),\ and$$

$$\mu \in (\mathcal{R}(Int(\bar{s}')))_{|\mathcal{V}(\bar{G}\sigma)} = (\mathcal{R}(Int(\bar{s}')))_{|\mathcal{V}(\bar{G}\sigma)}.$$

Then, by lemma 4.20 *(2)*, we have that there exists $\bar{s}'_1$ such that

$$< \bar{G}, \bar{s}'_1 > \in lfp(T_W),\ and$$

$$(\sigma\mu)_{|\mathcal{D}(\sigma)} \in (\mathcal{R}(\sigma.(Int(\bar{s}'_1))))_{|\mathcal{D}(\sigma)}. \tag{2}$$

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

$$< \bar{A}'\sigma\mu, \bar{s}'' > \in lfp(T_W),\ and$$

$$\vartheta' \in (\mathcal{R}(Int(\bar{s}'')))_{|\mathcal{V}(\bar{A}'\sigma\mu)} = (\mathcal{R}(Int(\bar{s}'')))_{|\mathcal{V}(\bar{A}'\sigma\mu)}. \tag{3}$$

By lemma 4.20 *(2)*, there exists $\bar{s}''_1$ such that

$$< \bar{A}'\sigma\mu, \bar{s}''_1 > \in lfp(T_W),\ and$$

$$(\mathcal{R}((\sigma\mu).(Int(\bar{s}''_1))))_{|\mathcal{D}(\sigma\mu)} = (\sigma\mu\mathcal{R}(Int(\bar{s}'')))_{|\mathcal{D}(\sigma\mu)}. \tag{4}$$

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

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

we have

$$< A_i, s_i > \in lfp(T_W).$$

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

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

Therefore

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

$\square$

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

Theorem 4.23 **(Completeness)** Let W be a Parlog program and let $\bar{A}$ be a sequence of atoms. Then

$$\{\vartheta \mid \exists \bar{s} \text{ locally independent on } \bar{A} : < \bar{A}, \bar{s} > \in \mathcal{F}(W) \text{ and } \vartheta \in \mathcal{R}(\| \, (\bar{s}))_{|\mathcal{V}(\bar{A})}\}$$

$$\subseteq$$

$$\mathcal{O}_1(\leftarrow \bar{A}).$$

Proof Let $\bar{s} = s_1, \ldots, s_n$. We prove the theorem by induction on the lenght $\#(\bar{s})$ 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 ϑ, and $\vartheta = (\mathcal{E}(\vartheta'))_{|\mathcal{V}(\bar{A})} = \vartheta'_{|\mathcal{V}(\bar{A})}$. Then, there exists a clause of the form $H \leftarrow \, | \in W_{\mathcal{V}(A)}$ and $\vartheta' \in mgu_i(A^-, H)$ holds. Then, since $\mathcal{E}(\vartheta') \neq \emptyset$, we have $\vartheta'_{|\mathcal{V}(\bar{A})} = \epsilon$. Therefore

$$< \leftarrow A, \epsilon > \rightarrow^1 < \square, \vartheta' > .$$

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

   $$< \leftarrow \bar{A}, \epsilon > \rightarrow^1 < \leftarrow (A_1, \ldots, A_{i-1}, \bar{B}, A_{i+1}, \ldots, A_n), \sigma >$$

   and

   $$\exists \bar{r} :< \bar{B}, \bar{r} > \in lfp(T_W).$$

   By lemma 4.20 *(1)*, there exist $\bar{s}', s'', \bar{s}''$ such that

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

   (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 $H \leftarrow \bar{G}|\bar{B} \in W_{\mathcal{V}(A_i)}$ such that

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

   From lemma 4.20 *(1)* it follows that there exists $\bar{r}'$ such that

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

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

$$<\leftarrow \bar{G}, \sigma > \longrightarrow^{k} < \Box, \sigma\tau >$$

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

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

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

$$<\leftarrow \bar{A}, \epsilon > \longrightarrow^{k+1} <\leftarrow (A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n, \bar{B}), \sigma\tau > .$$

The rest follows as in case (1).

□

Example 4.24

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 $< p(x), s > \in lfp(T)$, are those of the form $s = [\{y/x^-\}.\{y/a\}]_{\{x^-\}}$. We have: $\mathcal{R}(s) = disann_{\{x^-\}}(\mathcal{E}(\{y/x^-\} \hat{\circ} \{y/a\})) = disann_{\{x^-\}}(\mathcal{E}(\{\{x^-/a, y/a\}\})) = \emptyset$, 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: $\mathcal{R}(s) = \mathcal{R}(disann_{\{x^-\}}(\mathcal{E}(\{y/x^-\})) \hat{\circ} \{y/a\}) = \mathcal{R}(\{y/x\} \hat{\circ} \{y/a\}) = \{\{x/a, y/a\}\}$, 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 4.25

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 $< p(x,y), s_1 >, < q(x,y), s_2 > \in lfp(T)$, for $s_1 = [\{x^-/a\}]_{\{x^-\}}.\{y/b\}$ and $s_2 = [\{y^-/b\}]_{\{y^-\}}.\{x/a\}$. For all the possible interleavings $s \in s_1 \parallel s_2$, we get $\mathcal{R}(s) = \emptyset$. 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 $< p(x,y), s_1 >, < q(x,y), s_2 > \in lfp(T)$, for $s_1 = [\{z/x^-\}]_{\{x^-\}}.\{y/b\}.[\{z^-/a\}]_{\{z^-\}}$ and $s_2 = [\{y^-/b\}]_{\{y^-\}}.\{x/a\}$. We have $s = [\{z/x^-\}]_{\{x^-\}}.\{y/b\}.[\{y^-/b\}]_{\{y^-\}}.\{x/a\}.[\{z^-/a\}]_{\{z^-\}} \in s_1 \parallel s_2$ and $\{x/a, y/b, z/a\} \in \mathcal{R}(s)$. 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 [BZ82], 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 [BZ82] and [AR89], it is described how to solve 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

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

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 Γ is given by

$$(\alpha \in)\ \Gamma = V \cup V^{[\,]}$$

with

$$(f \in)\ V = \mathit{Subst} \rightarrow \mathit{Subst}_\delta, \quad V^{[\,]} = \{[f] : f \in V\}.$$

Here $\mathit{Subst}_\delta = \mathit{Subst} \cup \{\delta\}$ and δ 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 $\{< \alpha_i, p_i >: i \in I\}$, for some index set I. In that case, p has the choice among the steps $< \alpha_i, p_i >$. Each step consists of some action α_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 in the semantics of Parlog 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 lenght 1.) A typical example of a process is

$$\begin{aligned} p = \{< f_1, \{ \ & < [f_2], \{< [f_3], p_0 >\} >, \\ & < f_4, \{< [f_5], p_0, < [f_6], p_0 >\} >\} >\}. \end{aligned}$$

We shall use the following semantic operators.

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

1. $p_0; q = q,\ p; q = \{< \alpha, p'; q > \mid \ < \alpha, p' > \in p\}.\ ; q = \{< \alpha, p'; q > \mid \ < \alpha, p' > \in p\}.$
2. $p_0 \parallel q = q \parallel p_0 = q,$
   $p \parallel q = p \mathbin{|\!\lfloor} q \cup q \mathbin{|\!\lfloor} p,$
   $p \mathbin{|\!\lfloor} q = \{< \alpha, p' > \mathbin{|\!\lfloor} q \mid \ < \alpha, p' > \in p\},$
   $< f, p' > \mathbin{|\!\lfloor} q = < f, p' \mathbin{|\!\lfloor} q >,\ < [f], p' > \mathbin{|\!\lfloor} q = < [f], p' \parallel q >.$
3. $$\begin{aligned} \mathit{int}(p_0) &= p_0 \\ \mathit{int}(p) &= \{< f, \mathit{int}(p') > \mid (< f, p' > \in p \vee < [f], p' > \in p) \wedge p' \neq p_0\} \\ &\cup \ \{< [f], p_0 > \mid \ < f, p_0 > \in p \vee < [f], p_0 > \in p\}. \end{aligned}$$

□

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 [ ], we have an interleaving point. This explains the definition 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: $< f, p' > \|\!\lfloor\, q = < f, p' \| q >$. If, on the other hand, an interleaving point is reached then we switch back to the ordinary merge again: $< [f], p' > \|\!\lfloor\, q = < [f], p' \| q >$.

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 ready for the definition of a denotational semantics for Parlog. Let W be a fixed program.

Definition 5.3 We define $\mathcal{D} : \mathcal{P}(Var) \to Goal \to P$ as follows:

1. $\mathcal{D}\llbracket X \rrbracket \llbracket \square \rrbracket = p_0$
2. $$\mathcal{D}\llbracket X \rrbracket \llbracket \leftarrow A \rrbracket = \begin{array}{l} \bigcup\{int(\{< f_i(A,H,X), \mathcal{D}\llbracket X \cup invar(A) \rrbracket \llbracket \leftarrow \bar{G} \rrbracket >\}); \\ (\mathcal{D}\llbracket X \rrbracket \llbracket outunif(A,H), \bar{B} \rrbracket) : H \leftarrow \bar{G} | \bar{B} \in W\} \end{array}$$
   with
   $$f_i(A,H,X) = \lambda\vartheta. \begin{cases} \vartheta mgu_i(A\vartheta, H) & \text{if } mgu_i(A\vartheta, H)\downarrow \text{ and } mgu_i(A\vartheta, H)|_{X\vartheta \cup invar(A\vartheta)} = \epsilon \\ \delta & \text{otherwise} \end{cases}$$
   (Here $X\vartheta = \bigcup\{var(\vartheta(x)) : x \in X\}$.)
3. $\mathcal{D}\llbracket X \rrbracket \llbracket \leftarrow outunif(A,H) \rrbracket = \{< f_o(A,H,X), p_0 >\}$
   with
   $$f_o(A,H,X) = \lambda\vartheta. \begin{cases} \vartheta mgu_o(A\vartheta, H) & \text{if } mgu_o(A\vartheta, H)\downarrow \\ \delta & \text{otherwise} \end{cases}$$
4. $\mathcal{D}\llbracket \leftarrow \bar{A}, \bar{B} \rrbracket = \mathcal{D}\llbracket \leftarrow \bar{A} \rrbracket \| \mathcal{D}\llbracket \leftarrow \bar{B} \rrbracket$.

□

(Note that the definition of $\mathcal{D}$ is recursive; like the semantic operators, it can be given as the fixed point of a contraction.) The first argument X of $\mathcal{D}$ 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 $\mathcal{D}$). It is used in the definitions of f_i and f_o. In clause 2, X is changed into $X \cup invar(A)$, 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 $invar(A)$ 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 is 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 $\mathcal{D}$ with respect to $\mathcal{O}_2$

We shall relate $\mathcal{O}_2$ and $\mathcal{D}$ via a function $yield : P \rightarrow Subst \rightarrow M_2$. 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 Φ, which will allow for an easy equivalence proof. First we turn M_2 into a complete metric space.

Definition 6.1 We define $M_2 = \mathcal{P}_c(Subst_\delta^\infty)$, where $\mathcal{P}_c$ 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 $Subst_\delta^\infty$. □

Now we can define a contraction

$$\Phi : (Goal \rightarrow Subst \rightarrow M_2) \rightarrow (Goal \rightarrow Subst \rightarrow M_2)$$

by

$$\Phi(F)[\![\leftarrow \bar{A}]\!](\vartheta) = \bigcup\{\vartheta' \cdot F[\![\leftarrow \bar{A}']\!](\vartheta') : <\leftarrow \bar{A}, \vartheta > \rightarrow < \rightarrow \bar{A}', \vartheta' >\}$$

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 $Goal \rightarrow Subst \rightarrow M_2$ 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 $yield : P \rightarrow Subst \rightarrow M_2$ be given by

$$\begin{aligned} yield(p_0)(\vartheta) &= \{e\} \text{ (the empty sequence)} \\ yield(p)(\vartheta) &= \bigcup_\delta\{\vartheta' \cdot yield(p_n)(\vartheta') : < f_1, p_1 > \in p \wedge \cdots \wedge < f_{n-1}, p_{n-1} > \in p_{n-2} \wedge \\ &\quad < [f_n], p_n > \in p_{n-1} \wedge (f_n \circ \cdots \circ f_1)(\vartheta) = \vartheta'\} \end{aligned}$$

□

(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 $\{p : \exists \bar{A}, X (p = \mathcal{D}[\![X]\!][\![\leftarrow \bar{A}]\!])\}$ 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

$$\begin{aligned} \bigcup\nolimits_\delta X &= \bigcup X \setminus \{\delta\} && \text{if } \bigcup X \setminus \{\delta\} \neq \emptyset \\ &= \{\delta\} && \text{otherwise.} \end{aligned}$$

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 ϑ and state transformation f the next state by applying f to ϑ. 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

$$< f_1, p_1 >, \ldots, < f_{n-1}, p_{n-1} >, < [f_n], p_n >$$

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 $\mathcal{D}$ and the operational semantics $\mathcal{O}_2$. In the theorem below we shall allow ourselves the follwing abuse of language by writing $yield \circ \mathcal{D}$ for

$$\lambda\vartheta.\ \lambda \leftarrow \bar{A}.\ yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{A}\rrbracket)(\vartheta)$$

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

Proof
We show that for all ϑ, ϑ', $\bar{A}$, $\bar{A}'$

$$\begin{gathered} <\leftarrow \bar{A}, \vartheta > \rightarrow <\leftarrow \bar{A}', \vartheta' > \\ (1) \Leftrightarrow \\ yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{A}\rrbracket)(\vartheta) \supseteq \vartheta' \cdot yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{A}'\rrbracket)(\vartheta') \end{gathered}$$

From this it follows that

$$(2)\ \Phi(yield \circ \mathcal{D})\llbracket\leftarrow \bar{A}\rrbracket(\vartheta) = yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{A}\rrbracket)(\vartheta)$$

since

$$\begin{array}{rcl} \Phi(yield \circ \mathcal{D})\llbracket\leftarrow \bar{A}\rrbracket(\vartheta) & = & [\ \text{definition } \Phi\ ] \\ & & \bigcup_{\delta}\{\vartheta' \cdot yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{A}'\rrbracket)(\vartheta') : <\leftarrow \bar{A}, \vartheta > \rightarrow <\leftarrow \bar{A}', \vartheta' >\} \\ & \subseteq & [\ \text{by } (1)\ ] \\ & & yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{A}\rrbracket)(\vartheta) \end{array}$$

and

$$yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{A}\rrbracket)(\vartheta) \subseteq \bigcup_{\delta}\{\vartheta' \cdot yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{A}'\rrbracket)(\vartheta') : <\leftarrow \bar{A}, \vartheta > \rightarrow <\leftarrow \bar{A}', \vartheta' >\}$$

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

$$(3)\ \vartheta' \cdot yield(p)(\vartheta') \subseteq yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{A}\rrbracket)(\vartheta)\ \Rightarrow\ \exists \leftarrow \bar{A}' : p = \mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{A}'\rrbracket$$

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 $yield \circ \mathcal{D}$ is a fixed point of Φ. Both fixed points have to be equal by Banach's Theorem.

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

Case 1: $\Box$, trivial.

Case 2: $\leftarrow A$
By definition of $\to$ we have $<\leftarrow A, \vartheta> \to <\leftarrow \bar{A}, \vartheta'>$ if and only if there exist $H \leftarrow \bar{G}|\bar{B}$ and $\bar{\vartheta}$ such that

$$(4)\ <\leftarrow \bar{G}, mgu_i(A\vartheta, H)> \overset{*}{\to} <\Box, \bar{\vartheta}>$$

and

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

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

$$\vartheta_1 \cdot \cdots \cdot \vartheta_n \cdot \bar{\vartheta} \cdot yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\Box\rrbracket)(\bar{\vartheta}) \subseteq yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{G}\rrbracket)(mgu_i(A\vartheta, H))$$

or, since $yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\Box\rrbracket)(mgu_i(A\vartheta, H)) = \{e\}$,

$$\vartheta_1 \cdot \cdots \cdot \vartheta_n \cdot \bar{\vartheta} \in yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{G}\rrbracket)(mgu_i(A\vartheta, H))$$

Now we have

$$\begin{array}{l}
\vartheta_1 \cdot \cdots \cdot \vartheta_n \cdot \bar{\vartheta} \in yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{G}\rrbracket)(mgu_i(A\vartheta, H)) \\
\Leftrightarrow\ [\text{ definitions } yield \text{ and } int\ ] \\
\bar{\vartheta} \in yield(int(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow \bar{G}\rrbracket))(mgu_i(A\vartheta, H)) \\
\Leftrightarrow\ [\text{ using that } \bar{\vartheta}|_{invar(A)} = \epsilon\ ] \\
\bar{\vartheta} \in yield(int(\mathcal{D}\llbracket invar(A)\rrbracket\llbracket\leftarrow \bar{G}\rrbracket))(mgu_i(A\vartheta, H)) \\
\Leftrightarrow \\
\bar{\vartheta} \in yield(int(\mathcal{D}\llbracket invar(A)\rrbracket\llbracket\leftarrow \bar{G}\rrbracket))(\vartheta mgu_i(A\vartheta, H)) \\
\Leftrightarrow\ [\text{ definitions } yield,\ f_i(A, H, \emptyset);\text{ recall that } \vartheta' = \vartheta\bar{\vartheta}\ ] \\
(5)\ \vartheta' \in yield(int(\{< f_i(A, H, \emptyset), \mathcal{D}\llbracket invar(A)\rrbracket\llbracket\leftarrow \bar{G}\rrbracket >\}))(\vartheta)
\end{array}$$

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

$$\begin{array}{rcl}
yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow A\rrbracket)(\vartheta) & = & yield(int(\{< f_i(A, H, \emptyset), \mathcal{D}\llbracket invar(A)\rrbracket\llbracket\leftarrow \bar{G}\rrbracket >\}); \\
 & & \mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow outunif(A, H), \bar{B}\rrbracket :\ H \leftarrow \bar{G}|\bar{B} \in W\})(\vartheta) \\
 & = & \bigcup\{\vartheta' \cdot yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow outunif(A, H), \bar{B}\rrbracket)(\vartheta') : \\
 & & \vartheta' \in yield(int(\{< f_i(A, H, \emptyset), \mathcal{D}\llbracket invar(A)\rrbracket\llbracket\leftarrow \bar{G}\rrbracket >\}))(\vartheta), \\
 & & H \leftarrow \bar{G}|\bar{B} \in W\}
\end{array}$$

(For the latter equality we use the fact that

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

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

$$\vartheta' \cdot yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow outunif(A, H), \bar{B}\rrbracket)(\vartheta') \subseteq yield(\mathcal{D}\llbracket\emptyset\rrbracket\llbracket\leftarrow A\rrbracket)(\vartheta)$$

Summarizing we have

$$<\leftarrow A, \vartheta> \rightarrow <(\leftarrow outunif(A,H), \bar{B}), \vartheta'>$$

$$\Leftrightarrow$$

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

which is what we wanted to show.

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

Case 4: $\leftarrow \bar{A}, \bar{B}$
We have $<\leftarrow \bar{A}, \bar{B}, \vartheta> \rightarrow <\leftarrow \bar{A}'', \vartheta'>$ if and only if

$$<\leftarrow \bar{A}, \vartheta> \rightarrow <\leftarrow \bar{A}', \vartheta'>, \quad \leftarrow \bar{A}'' = \leftarrow \bar{A}', \bar{B}$$

or

$$<\leftarrow \bar{B}, \vartheta> \rightarrow <\leftarrow \bar{B}', \vartheta'>, \quad \leftarrow \bar{A}'' = \leftarrow \bar{A}, \bar{B}'$$

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

$$<\leftarrow \bar{A}, \vartheta> \rightarrow <\rightarrow \bar{A}', \vartheta'>$$

$$\Leftrightarrow$$

$$(6)\ yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!])(\vartheta) \supseteq yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}']\!])(\vartheta')$$

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

$$\begin{array}{rcl} yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}, \bar{B}]\!])(\vartheta) & = & yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!] \parallel \mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{B}]\!])(\vartheta) \\ & \supseteq & yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!] \lfloor\!\lfloor \mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{B}]\!])(\vartheta) \\ & = & \bigcup_{\delta}\{\vartheta' \cdot yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}']\!] \parallel \mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{B}]\!])(\vartheta) : \\ & & \vartheta' \cdot yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}']\!])(\vartheta') \subseteq yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!])(\vartheta)\} \\ & = & \bigcup_{\delta}\{\vartheta' \cdot yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}', \bar{B}]\!])(\vartheta) : \\ & & \vartheta' \cdot yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}']\!])(\vartheta') \subseteq yield(\mathcal{D}[\![\emptyset]\!][\![\leftarrow \bar{A}]\!])(\vartheta)\} \end{array}$$

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

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

for all p, q and ϑ.) Thus we see that (6) above is equivalent with

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

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 PARLOG. Then, we have defined a denotational semantics, correct with respect to the operational one, that (compositionally) models also the finite failures and the infinite computations. Similar approaches can be taken for GHC (see [BKPR], [P] for the declarative part) and, we believe, for Concurrent Prolog.

If we compare the denotational semantics given here to the ones given in [BK88] and [BKPR88a], 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 [BKPR88a]), because the restrictions imposed on unifications by the mode declaration of a program are treated in the same way by both the denotational model 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.

References

[Apt87] K.R. Apt. *Introduction to logic programming.* Technical Report CS-R8741, Centre for Mathematics and Computer Science, Amsterdam, 1987. To appear as a chapter in Handbook of Theoretical Computer Science, North-Holland.

[AR89] P. America and J.J.M.M. Rutten. *Solving reflexive domain equations in a category of complete metric spaces.* Journal of Computer and System Sciences, Vol. 39, No. 3, 1989, pp. 343-375.

[Be86] L. Beckman, *Towards a Formal Semantics for Concurrent Logic Programming Languages*, Proc. of the Third International Conference on Logic Programming, Lecture Notes in Computer Science, Vol. 225, Springer, 1986, pp. 335-349.

[BK88] J.W. Bakker and J.N. Kok. *Uniform abstraction, atomicity and contractions in the comparative semantics of concurrent prolog.* Technical Report CS 8834, Centre for Mathematics and Computer Science, Amsterdam, 1988. Extended abstract appeared in proceedings Fifth Generation Computer Systems (FGCS 88), pp. 347-355.

[BKPR88a] 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.* Proceedings Mathematical Foundations of Computer Science (MFCS 89), Lecture Notes in Computer Science 379, pp. 165-177.

[BKPR88b] F.S. de Boer, J.N. Kok, C. Palamidessi, and J.J.M.M. Rutten. *Semantic models for a version of PARLOG*, Proceedings International Conference on Logic Programming (ICLP 89), MIT Press, 1989.

[BM87] J.W. de Bakker and J.-J.Ch. Meyer. *Metric semantics for concurrency.* BIT, 28, 1988, pp. 504-529.

[BZ82] J.W. de Bakker and J.I. Zucker. *Processes and the denotational semantics of concurrency.* Information and Control 54, 1982, pp. 70-120.

[CG85] K.L. Clark, S. Gregory, *Notes on the implementation of PARLOG,* Journal of Logic Programming 2(1), 1985, 17-42.

[CG86] K.L. Clark, S. Gregory, *PARLOG: Parallel programming in logic,* ACM Trans. Program. Lang. Syst. Vol. 8, 1, 1986, 1-49. Res. Report DOC 84/4, Dept. of Computing, Imperial College, London,1984.

[Ede85] E. Eder. *Properties of substitutions and unifications.* Journal Symbolic Computation 1, 1985, pp. 31-46.

[vEK76] M.H. van Emden and R.A. Kowalski. *The semantics of predicate logic as a programming language.* Journal of the ACM 23(4), 1976, 733-742.

[FL88] M. Falaschi, G. Levi, *Finite Failures and Partial Computations in Concurrent Logic Languages*, Proceedings Fifth Generation Computer Systems (FGCS 88), 1988, pp. 364-373.

[FLPM88a] M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli. *Declarative modeling of the operational behaviour of logic languages.* Theoretical Computer Science. To appear.

[FLPM88b] M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli. *A new declarative semantics for logic languages.* In Proceedings Conference and Symposium on Logic Programming, MIT press, 1988, pp. 993-1005.

[GCLS88] R. Gerth, M. Codish, Y. Lichtenstein, and E. Shapiro. *Fully abstract denotational semantics for concurrent prolog.* In Proc. Logic In Computer Science, 1988, pp. 320-335.

[Gre87] S. Gregory. *Parallel logic programming in PARLOG.* International Series in Logic Programming, Addison-Wesley, 1987.

[Hu] G. Huet *Resolution d'Equations dans des Langages d'Order 1, 2,. . . ,ω.* These d'Etat, Univ. Paris VII.

[HP79] M. Hennessy and G.D. Plotkin. *Full abstraction for a simple parallel programming language.* In J. Becvar, editor, Proceedings 8th MFCS, Lecture Notes in Computer Science 74, Springer Verlag, 1979, pp. 108-120.

[KR88] J.N. Kok and J.J.M.M. Rutten. *Contractions in comparing concurrency semantics.* In Proceedings 15th ICALP, Tampere, Lecture Notes in Computer Science 317, Springer Verlag, 1988, 317-332.

[LMM88] J.-L. Lassez, M.J. Maher, and K. Marriot. *Unification revisited.* In J. Minker, editor, Foundations of deductive databases and logic programming, Morgan Kaufmann, Los Altos, 1988.

[LP85] G. Levi and C. Palamidessi. *The declarative semantics of logical read-only variables.* In Proc. Symp. on Logic Programming, IEEE Comp. Society Press, 1985, pp. 128-137.

[LP87] G. Levi and C. Palamidessi. *An approach to the declarative semantics of synchronization in logic languages.* In Proc. 4th Int. Conference on Logic Programming, 1987, 877-893.

[M88] M. Murakami. *A New Declarative Semantics of Parallel Logic Programs with Perpetual Processes.* Proceedings Fifth Generation Computer Systems (FGCS 88), 1988, pp. 374-381.

[L88] G. Levi. *Models, unfolding rules and fixed point semantics.* Proc. Symp. on Logic Programming, 1988, pp. 1649-1665.

[Pal88] C. Palamidessi. *A fixpoint semantics for Guarded Horn Clauses.* Technical Report CS-R8833, Centre for Mathematics and Computer Science, Amsterdam, 1988.

[Sar85] V.A. Saraswat: *Partial Correctness Semantics for CP(↓, |, &).* Proc. of the Conference on Foundations of Software Computing and Theoretical Computer Science, LNCS 206, 1985, pp. 347-368.

[Sar87a] V.A. Saraswat: *The concurrent logic programming language CP: definition and operational semantics,* in: Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages, Munich, West Germany, January 21-23, 1987, pp. 49-62.

[Sar87b] V.A. Saraswat. *GHC: operational semantics, problems and relationship with* cp(↓, |). In IEEE international symposium on logic programming, 1987, pp. 347-358.

[Sh83] E.Y. Shapiro. *A subset of concurrent prolog and its interpreter.* Tech. Report TR-003, ICOT, Tokyo, 1983.

[Sh88] E.Y. Shapiro. *Concurrent Prolog: Collected Papers.* Vol. 1-2 MTI press, 1988.

[TF86] A. Takeuchi and K. Furukawa. *Parallel Logic Programming Languages.* Proc. 1986 Int. Conference on Logic Programming, LNCS 225, London, 1986, pp. 242-254.

[U85] K. Ueda. *Guarded Horn Clauses.* Technical Report TR-103, ICOT, Tokyo, 1985. Revised in 1986. A revised version is in Proc. Logic Programming 1985, pp. 168-179. Also in E.Y. Shapiro, editor, *Concurrent Prolog, Collected Papers.*, chapter 4.

[U88] K. Ueda. *Guarded Horn Clauses, A Parallel Logic Programming Language with the Concept of a Guard.* Technical Report TR-208, ICOT, Tokyo, 1986. Revised in 1987. Also in Proc. Programming of Future Generation Computers, North Holland, 1988, pp. 441-456.