Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

C. Palamidessi

A fixpoint semantics for guarded horn clauses

Computer Science/Department of Software Technology Report CS-R8833 September

Centrum voor Wiskunde en Informatica
Amsterdam

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 Pure Research (Z.W.O.).

69D13,69D41,69F12,69F32

Copyright © Stichting Mathematisch Centrum, Amsterdam

A Fixpoint Semantics for Guarded Horn Clauses *

Catuscia Palamidessi

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

Dipartimento di Informatica,
Università di Pisa,
Corso Italia 40, 56100 Pisa, Italy

Abstract

This paper gives a fixpoint characterization of the success set of Guarded Horn Clauses programs. The notions of substitution and unification, on which the declarative semantics of pure logic programs is based, are extended in order to deal with the synchronization mechanisms of Guarded Horn Clauses. The effectiveness of the new notions is shown by defining an algorithm that computes the most general unifier. Also the *immediate consequence operator*, used for defining the fixpoint semantics of pure logic programs, is extended consequently. Finally, the equivalence of the new fixpoint semantics with the operational semantics of Guarded Horn Clauses is proved.

Key words and phrases: concurrent logic languages, unification, input-mode constraint, fixpoint semantics, operational semantics, soundness, completeness.
1985 Mathematics Subject Classification: 68Q55, 68Q10.
1987 Computing Reviews Categories: D.1.3, D.3.1, F.1.2, F.3.2.

1 Introduction.

Logic languages, i.e. languages based on the Horn Clause Logic (HCL) [Ll], have become very popular in the last few years, mainly because of their high level nature. In fact, programs written in these formalisms are based on assertions which have a declarative interpretation. In other words, they can be seen as a formal description of the problem to be solved and can be understood without any reference to the behaviour of any particular machine. Moreover, they have the advantage of being founded on rigorous and well-established mathematical theories, that make easier the task of defining their formal semantics, of developing methods for correctness proofs, etc.

Another, increasingly important, reason of interest in these languages is due to the advent of VLSI technologies which has made possible the development of highly parallel computer architectures. This has caused a gap between hardware and software, since parallel programming is much more difficult than sequential programming. It is therefore necessary the

*This work was carried out during the stay of the author at the CWI. Visiting the CWI was made possible by support of the Italian Consiglio Nazionale delle Ricerche (CNR)

develop suitable software tools, and, in particular "good" languages for parallel programming. Due to their semantics clearness, lack of side effects, and high level constructs, the class of logic languages has been regarded as a suitable candidate to fully utilize the power of parallel architectures. In particular, it has been claimed that these languages are inherently parallel, i.e. that their operational semantics leads to a parallel computational model in a very elegant way. Moreover it has been claimed that their declarative nature facilitates the definition of parallel algorithms by the programmer.

Until now, these claims have been not completely realized in the practice. Moreover, they seem to be a little bit in conflict with respect to each other. In the parallel model of pure logic languages several problems arise, mainly due to the environment management. In particular, the unconstrained AND-OR parallelism is definitely too little efficient, and this makes necessary the introduction of some control mechanisms. Moreover, pure HCL languages are not expressive enough to model the actual aspects of concurrent systems, such as real time, synchronization, communication. Shared variables can be seen as communication channels between processes, but they are still a too weak notion.

Most of the concurrency-oriented extensions of logic languages are based on adding constraints to the unification mechanism (input-mode constraints). In practice, some of the variables occurring in the goal can be prevented from getting bound during the unification (i.e. from 'producing' values). In this way, the process (atom) in which such a variable occurs, can be forced to wait ('suspension rule') until other processes in the goal have made available the values required by that variable (that can therefore be seen as a 'consumer'). In this way, the communication channels can be seen as directed. Moreover, the suspension rule provides a synchronization mechanism. The most famous representatives of this class of languages are Concurrent Prolog (CP) [Sh1], [Sh2], PARLOG [CG] and Guarded Horn Clauses (GHC) [Ue].

Most of the efforts about these languages have been devoted to the solution of the efficiency and implementation problems mentioned before. For instance, the definition of PARLOG has quite been changed from its original version. Very little, on the contrary, has been done in the direction of preserving the clear semantics of their origins. The traditional semantics of HCL, in fact, is not suitable anymore to describe these languages, because of all the extra-logical features that have been added. In particular, it is quite hard to say whether or not these languages can still be considered declarative. It is worth noting that the semantics that have been developed for these languages are either in operational style, based on transition systems, [Sa], [Sa1], [Be], [BK], or in denotational style, based on metric spaces [BK], [K]. Both of these approach are based on the traditional techniques for the imperative languages.

The only declarative approaches to the semantics of these languages have been investigated in [LP1], [LP2] and [Le]. However, in all these works only a variant of these languages has been modeled. Namely, the *deadlocks* (situations in which two or more processes are mutually waiting for each other) cannot be described by these approaches.

On the other side, a declarative characterization of the real languages could be helpful to understand the difference that the new mechanisms introduce with respect to the pure versions.

The present paper is a first attempt to define a correct declarative semantics for these languages. We focus the attention on GHC, but we think that the same methods can also be applied to CP and PARLOG. We define a fixpoint semantics, i.e. a semantics based on a fixpoint construction, by making use of an operator that works on the Herbrand interpretations. This approach is usually called declarative because the definition of the operator is strictly related to the notion of model. The basic ideas on which our construction is based are:

- The introduction of variables in the notion of Herbrand interpretation. This idea, already used in [LP2] and in [FLMP], allows to model the notions of 'value produced' and

'value consumed' by an atom. This is necessary to distinguish between programs that differ only for some input-mode constraint.

- The notion of annotation on variables, representing the input-mode constraints. This idea is in a certain sense symmetric to the one used in [LP2], where the annotations are on data structures. The annotations are declaratively characterized by extending the theory of substitution and unification, and can be effectively manipulated by an extended unification algorithm, also described in the paper.

- The notion of streams of substitutions. This concept represents the main difference with respect to [LP2], and has been introduced in order to deal with the deadlock problem mentioned above.

This paper is organized as follows. Section 2 recalls some basic notions about logic languages. Section 3 introduces the language GHC. Section 4 defines the basic mathematical tools, such as the new notions of substitution and unification. Finally, section 5 defines the fixpoint operator and shows the equivalence between the fixpoint semantics and the operational semantics of GHC.

2 Preliminaries

In this section we briefly recall some basic notions about logic programs, substitutions and unification. We will use mainly the same terminology and notations of [Ap], [Ll], [LMM], and [Ed], to which the reader is referred for a more detailed presentation of these topics.

Logic Programs. Let *Var* be a set of *variables*, with typical elements $x, y, z, \ldots$. Let *Cons* be a set of *constructors*, with typical elements $a, b, c, \ldots$ (constructors with 0 arguments, or *constants*), and $f, g, h, \ldots$ (constructors with one or more arguments). Let *Pred* be a set of *predicates*, with typical elements $p, q, r, \ldots$.

The set *Term*, with typical elements $t, u, \ldots$, is the set of all the *terms* built on *Var* and on *Cons*. Examples of terms are $f(a), f(x), g(f(a), f(x))$ etc.

The set *Atom*, with typical elements $A, B, H, G \ldots$, is the set of all the *atoms* (or *atomic formulas*) built on *Pred* and on *Term*. Examples of atoms are $p(a), p(x), q(f(a), f(x))$ etc.

A *definite clause* is a formula of the form

$$A \leftarrow B_1, \ldots, B_n \quad (n \geq 0)$$

where the intended meaning of "$\leftarrow$" is the right-to-left logic implication, and the intended meaning of "," is the logical conjunction. Clauses are seen as universally quantified.

A *goal statement* is a formula of the form

$$\leftarrow A_1, \ldots, A_m \quad (m \geq 0)$$

(the missing argument at the left of $\leftarrow$ is meant to be the logical value *false*). If $m = 0$, the goal is called *empty goal* and is denoted by $\Box$. Goal statements are also seen as universally quantified.

A Horn Clause Logic (HCL) program W is a (finite) set of definite clauses. Since a clause (as well as a goal statement) is a particular First Order Logic formula, HCL programs can be seen as a particular subset of the First Order Logic theories.

In the following, we use the abbreviations $\bar{A}, \bar{B}$ etc. to denote conjunctions of atoms, like $A_1, \ldots, A_m$ and $B_1, \ldots, B_n$. Then, for instance, $A \leftarrow \bar{B}$ represents a clause, and $\leftarrow \bar{A}$ represents a goal statement.

Substitutions. A substitution ϑ is a mapping from *Var* into *Term* such that $\mathcal{D}(\vartheta) = \{x \in Var \mid \vartheta(x) \neq x\}$ is finite. $\mathcal{D}(\vartheta)$ is called the domain of the substitution ϑ. We will use also the set-theoretic notation for ϑ, given by $\vartheta = \{x/t \mid x \in \mathcal{D}(\vartheta), \vartheta(x) = t\}$.

Let F be an expression (term, atom or clause). The set of variables occurring in F is denoted by $\mathcal{V}(F)$. The application $F\vartheta$ of ϑ to F is defined as the expression obtained by replacing each variable x in F by $\vartheta(x)$. $\mathcal{C}(\vartheta)$ (which we will improperly call the co-domain, or range of ϑ) is the set $\bigcup_{x \in \mathcal{D}(\vartheta)} \mathcal{V}(\vartheta(x))$.

A renaming ρ is any bijective substitution from *Var* to *Var*. If V is a set of variables, then $F\rho$ is a variant of a formula F with respect to V iff ρ is a renaming and $\mathcal{V}(F\rho) \cap V = \emptyset$. $F\rho$ is said to be a variant of F iff $F\rho$ is a variant of F with respect to $\mathcal{V}(F)$.

If W is a GHC program, and V is a set of variables, then W_V will denote all the possible *variants* of W with respect to V, i.e. the programs whose clauses are variants, with respect to V, of the clauses of W.

The composition $\vartheta\vartheta'$ of two substitutions ϑ and ϑ' is defined in the usual way, namely $(\vartheta\vartheta')(x) = (\vartheta(x))\vartheta'$. We recall that the composition is associative, the empty substitution ϵ is the neutral element, and for each renaming ρ there exists the inverse ρ^{-1}, i.e. $\rho\rho^{-1} = \rho^{-1}\rho = \epsilon$. Moreover, $F(\vartheta\vartheta') = (F\vartheta)\vartheta'$ holds.

A substitution ϑ is called idempotent iff $\vartheta\vartheta = \vartheta$ (or, equivalently, iff $\mathcal{D}(\vartheta) \cap \mathcal{C}(\vartheta) = \emptyset$). The pre-order relation $\leq$ on substitutions is defined by $\vartheta \leq \vartheta' \Leftrightarrow \exists \sigma : \vartheta\sigma = \vartheta'$, and the associated equivalence relation will be denoted by $\sim$.

The restriction $\vartheta_{|V}$ of ϑ to a set of variables V is the substitution $\vartheta_{|V}(x) = \vartheta(x)$ for $x \in V$ and $\vartheta_{|V}(x) = x$ otherwise.

Unification. Given a set of sets of terms M, a substitution ϑ is a unifier of M iff $\forall S \in M : \forall t, t' \in S : t\vartheta = t'\vartheta$ holds. It is well known that if there exists a unifier of M, then there exists an idempotent unifier of M (see, for instance, [Ap]). In order to simplify the discussion we will only consider, without loss of generality, the set of idempotent unifiers of M, denoted by $\mathcal{U}(M)$.

We associate with each substitution ϑ the set (of sets) $S(\vartheta) = \{\{x, t\} \mid x/t \in \vartheta\}$. The following proposition gives a useful characterization of the ordering relation, on idempotent substitutions, in terms of the solutions of the associated sets.

Proposition 2.1 *Let ϑ, ϑ' be idempotent substitutions. Then $\vartheta \leq \vartheta' \Leftrightarrow \mathcal{U}(S(\vartheta')) \subseteq \mathcal{U}(S(\vartheta))$.*

Proof First, we note that $\mu \in \mathcal{U}(S(\vartheta)) \Leftrightarrow \vartheta\mu = \mu$. Then, we have:

($\Leftarrow$) If ϑ' is idempotent, then $\vartheta' \in \mathcal{U}(S(\vartheta'))$. By hypothesis, $\vartheta' \in \mathcal{U}(S(\vartheta))$, and therefore $\vartheta\vartheta' = \vartheta'$.

($\Rightarrow$) Let σ be a substitution such that $\vartheta\sigma = \vartheta'$. Let $\mu \in \mathcal{U}(S(\vartheta'))$. Since ϑ is idempotent, we have $\vartheta\mu = \vartheta\vartheta'\mu = \vartheta\vartheta\sigma\mu = \vartheta\sigma\mu = \vartheta'\mu = \mu$, i.e., $\mu \in \mathcal{U}(S(\vartheta))$. □

The set of the most general unifiers of a set of sets of terms M is

$$mgu(M) = \{\vartheta \in \mathcal{U}(M) \mid \forall \vartheta' \in \mathcal{U}(M) : \vartheta \leq \vartheta'\}.$$

Note that, if ϑ is idempotent, then $\vartheta \in mgu(S(\vartheta))$ holds. The application $M\vartheta$ of a substitution ϑ to M is defined as the set (of sets) obtained by applying ϑ to all the terms occurring in the elements of M. In [Ed] the following basic property is shown (easily provable by the unification algorithm as defined, for instance, in [Ap]).

Proposition 2.2 *Let M_1, M_2 be sets of sets of terms. Let $\vartheta_1 \in mgu(M_1), \vartheta_2 \in mgu(M_2)$. Then*

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

where, for Θ *being a set of substitutions, the composition* $\vartheta\Theta$ *is defined as the set*

$$\{\vartheta' \mid \exists \vartheta'' \in \Theta : \vartheta' \in mgu(S(\vartheta\vartheta''))\}.$$

□

Corollary 2.3 *Let* ϑ_1, ϑ_2 *be idempotent substitutions. Then*

$$mgu(S(\vartheta_1) \cup S(\vartheta_2)) = \vartheta_1 mgu(S(\vartheta_2)\vartheta_1) = \vartheta_2 mgu(S(\vartheta_1)\vartheta_2).$$

Proof Since ϑ_1 is idempotent, $\vartheta_1 \in mgu(S(\vartheta_1))$. Analogously, $\vartheta_2 \in mgu(S(\vartheta_2))$. Now, assume $M_1 = S(\vartheta_1), M_2 = S(\vartheta_2)$ and apply proposition 2.2. □

Semantics of HCL. A very elegant formalization of the semantics of HCL was given by van Emden and Kowalski in [vEK]. HCL can be seen as a particular First Order Logic language, and therefore the *model-theoretic semantics*, developed by Tarski, for First Order Logic theories, applies also to HCL programs. Such a semantics is based on the notion of *validity*, i.e. *truth in all the models* of the programs. In particular, due to the *model-intersection* property of HCL programs, the set of *valid* atomic formulas can be characterized by a special model, namely the *minimal Herbrand model.*

The minimal Herbrand model can be obtained as the least fixpoint of a continuous transformation on Herbrand interpretations. Such a transformation is associated to the program, and can be seen as a sort of operator that gives all the *atomic immediate consequences* (with respect to the program) of a given set of (atomic) premises. This construction gives the so called *least fixpoint semantics.* Both these approaches (model theoretic and fixpoint), to the semantics of logic programs, are called *declarative.*

Finally, HCL programs have an *operational semantics* based on the notion of *refutation*, that can be seen also as a computational mechanism for the use of HCL as a programming language. Given a program W, and a goal $\leftarrow \bar{A}$, a *derivation step* consists of

- selecting an atom A_i in $\bar{A}$,
- selecting a clause $H \leftarrow \bar{B}$ in $W_{\mathcal{V}(\bar{A})}$,
- selecting a substitution ϑ in $mgu(A_i, H)$ (if any).

The derived goal is

$$\leftarrow (A_1, \ldots, A_{i-1}, \bar{B}, A_{i+1}, \ldots, A_n)\vartheta.$$

A refutation consists of the repeated application of the derivation step, until the empty goal is reached. The associated computed answer substitution is obtained as the composition of all the most general unifiers used in the derivation.

In [vEK] it is shown that all these three semantics are equivalent, with respect to the ground atomic formulas. Further developments have proved that the completeness of the operational semantics does not depend upon the choice of the atom in the refutation step (independence of the selection rule, [AvE]). Therefore, SLD refutation (refutation with respect to a fixed selection rule) can replace the refutation mechanism, with a considerable gain in efficiency.

Also the choice of ϑ in $mgu(A_i, H)$ is not relevant. This result, that has always been considered obvious, has been formally proved only recently in [LS].

Other investigations have been concerned with a more general declarative characterization of the operational aspects. The relation between the computed answer substitutions and the *correct answer substitutions* (i.e. the substitutions that make an atom true in all the models) is discussed in [Cl]. In [FLMP] a new declarative semantics is defined, and the full equivalence with the operational semantics (i.e. the equivalence with respect to all the atoms) is proved.

3 The language GHC and its operational semantics.

A Guarded Horn Clause (GHC) program [Ue] is a finite set of clauses of the form

$$H \leftarrow \bar{G} | \bar{B}$$

where H (head), $\bar{G}$ (guard) and $\bar{B}$ (body) are (sequences of) atomic formulas. The symbol "|" is called *commit operator.*

A GHC goal is a clause of the form

$$\leftarrow \bar{B}$$

The guard, the body and the goal can contain atoms of the form $t = u$, that are called *unification atoms* and play a special role. We will call the other ones *ordinary atoms.*

We give a description of the operational semantics based on the notion of refutation. The original definition given by Ueda [Ue] differs in that the proof method is explicitly AND-parallel (*parallel input resolution*). Anyway, in [Sa1] the two definition are shown equivalent (with respect to the success case).

Given a program W, a goal $\leftarrow \bar{A}$ has a *refutation* in W, with *computed answer substitution* ϑ iff

$$\leftarrow \bar{A} \stackrel{\vartheta'}{\longmapsto}{}^{*} \square \ (\textit{empty clause}) \ \textit{and} \ \vartheta = \vartheta'_{|\mathcal{V}(A)}$$

where $\bar{A} \stackrel{\vartheta'}{\longmapsto}{}^{*} \bar{A}'$ means that the goal $\leftarrow \bar{A}'$ is *derivable* from $\leftarrow \bar{A}$ in some steps. More formally, $\bar{A} \stackrel{\vartheta'}{\longmapsto}{}^{+} \bar{A}'$ holds iff there exists k such that $\bar{A} \stackrel{\vartheta'}{\longmapsto}{}^{k} \bar{A}'$, where $\stackrel{\vartheta'}{\longmapsto}{}^{k}$ is the k-steps derivation relation. Such a derivation is defined by the following rules, depending on the kind of the selected atom (unification atom or ordinary atom). Let $\bar{A} = A_1, \ldots, A_i, \ldots, A_n$, and let A_i be the selected atom. Then

$$\leftarrow \bar{A} \stackrel{\vartheta}{\longmapsto}{}^{k} \leftarrow \bar{A}'$$

if:

1. **(case: A_i is an ordinary atom)** the following conditions hold
   - $\exists H \leftarrow \bar{G} \mid \bar{B} \in W_{\mathcal{V}(A)}$
   - $\exists \vartheta' \in mgu(A_i, H)$
   - $\leftarrow \bar{G}\vartheta' \stackrel{\mu}{\longmapsto}{}^{k'} \square$
   - $(\vartheta'\mu)_{|\mathcal{V}(A)} = \epsilon$.

   In this case, we have
   - $\vartheta = \vartheta'\mu$
   - $k = k' + 1$
   - $\bar{A}' = (A_1, \ldots, A_{i-1}, \bar{B}, A_{i+1}, \ldots, A_n)\vartheta$.
2. **(case: A_i is a unification atom)** the following conditions hold
   - $\exists H \in \{x = x\}_{\mathcal{V}(A)}$
   - $\vartheta \in mgu(A_i, H)$.

   In this case, we have

- $k = 1$, and
- $\bar{A}' = (A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n)\vartheta$.

Note that, by definition of variant, the requirements $H \leftarrow \bar{G} \mid \bar{B} \in W_{\mathcal{V}(\bar{A})}$ (in the first case) and $H \in \{x = x\}_{\mathcal{V}(A)}$ (in the second case) ensure that the clauses used in the derivation do not share variables with the goal.

The basic synchronization mechanism of GHC (read-only constraint) is based on the condition $(\vartheta'\mu)_{|\mathcal{V}(\bar{A})} = \epsilon$. This condition represents the main difference from the operational semantics of HCL. It prevents the execution of atoms whose arguments are more general than the ones in the corresponding heads of the clauses. In practice, these atoms are obliged to *wait* until other atoms in the goal *produce* the necessary binding on the involved variables. Note that the unification atoms are the only ones that can produce bindings.

The derivation can be iterated on the new goal $\bar{A}'$. In general, if there exist $\bar{A}_1, \ldots, \bar{A}_n$ such that

$$\leftarrow \bar{A}_1 \overset{\vartheta_1 k_1}{\longmapsto} \leftarrow \bar{A}_2 \cdots \overset{\vartheta_{n-1} k_{n-1}}{\longmapsto} \leftarrow \bar{A}_n,$$

then we write

$$\leftarrow \bar{A}_1 \overset{\vartheta k}{\longmapsto} \leftarrow \bar{A}_n,$$

where

- $\vartheta = \vartheta_1 \cdots \vartheta_{n-1}$, and
- $k = k_1 + \cdots + k_{n-1}$.

Note that the superscript k on $\overset{\vartheta k}{\longmapsto}$ represents the total length of the refutation (included the number of steps performed in order to refute the guard).

The input-mode constraint, in the head-unification and in the refutation of the guard, forces a certain order in the execution of the atoms in a goal. Therefore, the SLD-resolution is not equivalent anymore to the resolution without any selection rule.

Other differences from the HCL programs rely on the presence of the commit operator, that transforms the *don't know* nondeterminism into *don't care* nondeterminism. However it influences only the *failure set*, whose analysis is out of the purposes of this paper. The interested reader can consult [FL] for the semantics of failure in GHC.

Example 3.1 *The following program realizes in GHC a merge on streams. The symbol cons represents the operation of prefixing a stream x with an element w, result $cons(w, x)$, and nil is the termination symbol. However, streams can be infinite.*

$$merge(x, y, z) \leftarrow x = cons(w, x') | z = cons(w, z'), merge(x', y, z').$$

$$merge(x, y, z) \leftarrow y = cons(w, y') | z = cons(w, z'), merge(x, y', z').$$

$$merge(x, y, z) \leftarrow x = nil | z = y.$$

$$merge(x, y, z) \leftarrow y = nil | z = x.$$

The initial goal is

$$\leftarrow p(x), q(y), merge(x, y, z), r(z).$$

Here $p(x)$ and $q(y)$ produce the streams input to the merge process, and $r(z)$ is the receiver of its output. □

4 A theory of substitutions and unification oriented to concurrency.

4.1 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 models* 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. With respect to the variables that are not shared, such a composition is simply the union of the respective bindings. With respect to the shared variables, ϑ_1 and ϑ_2 have to be consistent, and, if this is the case, the result is obtained as the minimal substitution that satisfies all the bindings (i.e. the minimal ϑ such that $\vartheta_1, \vartheta_2 \leq \vartheta$).

The basic requirement that this operation must satisfy is the independence of the order in which the atoms A_1 and A_2 are run. Namely, the result has to be equal to the one we get in a sequential execution of the same goal. By corollary 2.3, it turns out that 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. The consistency check corresponds to a verification that such a set is unifiable.

Definition 4.1 *Let ϑ_1, ϑ_2 be substitutions, and let Θ_1, Θ_2 be sets of substitutions. Then:*

1. $\vartheta_1 \hat{\circ} \vartheta_2 = \{\mu \mid \mu \text{ is idempotent}, \vartheta_1, \vartheta_2 \leq \mu \text{ and } \forall \sigma : \vartheta_1, \vartheta_2 \leq \sigma \Rightarrow \mu \leq \sigma\}$.
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. □

Proposition 4.2 *Let ϑ_1, ϑ_2 be idempotent substitutions. Then $\vartheta_1 \hat{\circ} \vartheta_2 = mgu(S(\vartheta_1) \cup S(\vartheta_2))$.*

Proof

($\subseteq$) Let $\mu \in \vartheta_1 \hat{\circ} \vartheta_2$. By proposition 2.1, $mgu(S(\mu)) = mgu(S(\vartheta_1) \cup S(\vartheta_2))$. Then, observe that, since μ is idempotent, $\mu \in mgu(S(\mu))$.

($\supseteq$) Let $\mu \in mgu(S(\vartheta_1) \cup S(\vartheta_2))$. Then, $\mathcal{U}(\mu) = \mathcal{U}(\vartheta_1) \cup S(\vartheta_2)$. Therefore, by proposition 2.1, $\mu \in \vartheta_1 \hat{\circ} \vartheta_2$.

□

Example 4.3

1. *Let $\vartheta_1 = \{x/f(y,a), z/g(b)\}$ and $\vartheta_2 = \{x/f(b,w), z/g(y)\}$. Then*
$$\vartheta_1 \hat{\circ} \vartheta_2 = \{\{x/f(b,a), z/g(b), y/b, w/a\}\}.$$
2. *Let ϑ_1 as before, and $\vartheta_2 = \{x/f(a,w), z/g(y)\}$. Then*
$$\vartheta_1 \hat{\circ} \vartheta_2 = \emptyset.$$

□

The following proposition shows that $\hat{\circ}$ is associative and commutative and (under a certain condition) distributes over the standard composition.

Proposition 4.4 *Let $\vartheta_1, \vartheta_2, \vartheta_3$ be substitutions. Then*

1. $\vartheta_1 \hat{\circ} \vartheta_2 = \vartheta_2 \hat{\circ} \vartheta_1$.
2. $(\vartheta_1 \hat{\circ} \vartheta_2) \hat{\circ} \vartheta_3 = \vartheta_1 \hat{\circ} (\vartheta_2 \hat{\circ} \vartheta_3)$.
3. *If ϑ_1 is idempotent, and $\mathcal{D}(\vartheta_1) \cap (\mathcal{D}(\vartheta_2) \cup \mathcal{D}(\vartheta_3) \cup \mathcal{C}(\vartheta_2) \cup \mathcal{C}(\vartheta_3)) = \emptyset$, then $\vartheta_1(\vartheta_2 \hat{\circ} \vartheta_3) = (\vartheta_1\vartheta_2) \hat{\circ} (\vartheta_1\vartheta_3)$.*

Proof *(1)* follows immediately from definition 4.1(*1*). For *(2)* it is sufficient to note that if $\vartheta \in \vartheta_1 \hat{\circ} \vartheta_2 = mgu(\mathcal{S}(\vartheta_1) \cup \mathcal{S}(\vartheta_2))$, then $\mathcal{S}(\vartheta)$ and $\mathcal{S}(\vartheta_1) \cup \mathcal{S}(\vartheta_2)$ are equivalent, in the sense that they have the same unifiers (see, for instance, [Ap]). Therefore

$$(\vartheta_1 \hat{\circ} \vartheta_2) \hat{\circ} \vartheta_3 = mgu(\mathcal{S}(\vartheta_1) \cup \mathcal{S}(\vartheta_2) \cup \mathcal{S}(\vartheta_3)) = \vartheta_1 \hat{\circ} (\vartheta_2 \hat{\circ} \vartheta_3).$$

For *(3)* we have:

$$\vartheta_1(\vartheta_2 \hat{\circ} \vartheta_3) =$$

$$\vartheta_1 mgu(\mathcal{S}(\vartheta_2) \cup \mathcal{S}(\vartheta_3)) = \text{ (by the hypotheses)}$$

$$\vartheta_1 mgu(\mathcal{S}(\vartheta_2)\vartheta_1 \cup \mathcal{S}(\vartheta_3)\vartheta_1) = \text{ (by corollary 2.3)}$$

$$mgu(\mathcal{S}(\vartheta_1) \cup \mathcal{S}(\vartheta_2) \cup \mathcal{S}(\vartheta_3)) = \text{ (by the hypotheses)}$$

$$mgu(\mathcal{S}(\vartheta_1\vartheta_2) \cup \mathcal{S}(\vartheta_1\vartheta_3)) =$$

$$(\vartheta_1\vartheta_2) \hat{\circ} (\vartheta_1\vartheta_3).$$

□

Similar properties hold for the sets of substitutions, which are formulated in the following corollary.

Corollary 4.5 *Let $\Theta_1, \Theta_2, \Theta_3$ be sets of substitutions, and let ϑ be an idempotent substitution. Then*

1. $\Theta_1 \hat{\circ} \Theta_2 = \Theta_2 \hat{\circ} \Theta_1$,
2. $(\Theta_1 \hat{\circ} \Theta_2) \hat{\circ} \Theta_3 = \Theta_1 \hat{\circ} (\Theta_2 \hat{\circ} \Theta_3)$,
3. *If $\mathcal{D}(\vartheta) \cap (\mathcal{D}(\Theta_1) \cup \mathcal{D}(\Theta_2) \cup \mathcal{C}(\Theta_1) \cup \mathcal{C}(\Theta_2)) = \emptyset$, then $\vartheta(\Theta_1 \hat{\circ} \Theta_2) = (\vartheta\Theta_1) \hat{\circ} (\vartheta\Theta_2)$,*

where the notions of domain and co-domain of a set of substitution are the natural extension of the corresponding notions on substitutions.

Proof Immediate from proposition 4.4 and definition 4.1(*2*). □

The associative property allows us to omit parentheses, and we will simply write $\vartheta_1 \hat{\circ} \vartheta_2 \hat{\circ} \vartheta_3 \hat{\circ} ...$ (and $\Theta_1 \hat{\circ} \Theta_2 \hat{\circ} \Theta_3 \hat{\circ} ...$).

Remark 4.6

1. *The empty set is the null element of the parallel composition (on sets), namely:*

   $$\Theta \hat{\circ} \emptyset = \emptyset \hat{\circ} \Theta = \emptyset.$$

2. *If Θ is* closed, *namely $\forall \vartheta \in \Theta : mgu(\mathcal{S}(\vartheta)) \subseteq \Theta$, then Θ is* idempotent *with respect to the parallel composition, i.e.*

   $$\Theta = \Theta \hat{\circ} \Theta.$$

Proof Immediate. □

4.2 Annotated variables.

In order to model the synchronization mechanism of GHC (the input-mode constraints), we introduce the notion of *annotated variable*. Intuitively, the annotation on a variable means that such a variable cannot be bound, during the derivation step, if it occurs in the selected atom. 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. From a declarative point of view, the difference between producing and receiving a binding can be modeled by introducing an asymmetry in the definition of the application of a substitution to a term: bindings on not-annotated variables (produced bindings) can instantiate the annotated variables, but bindings on annotated variables (consumed bindings) cannot instantiate the not-annotated variables.

We will denote the set of the annotated variables by Var^-. The elements of Var, still denoted by $x, y, z, \ldots$ will be called *positive variables*. The elements of Var^-, denoted by $x^-y^-, z^-, \ldots$, are seen as the annotated counterpart of $x, y, z, \ldots$, and will be called *negative variables*. From a mathematical point of view, we can consider Var and Var^- as two disjoint sets, and "$^-$" as a bijective mapping

$$^- : Var \rightarrow Var^-.$$

The set of terms $Term$ is extended on the new set of variables $Var \cup Var^-$, and we will call *positive terms* the ones in which there occur only positive variables. We extend the mapping $^-$ to the positive terms. If $V \subseteq Var$, $t^{|V^-}$ is the term obtained by replacing in t every variable $x \in V$ by x^-. $t^{|Var^-}$ will be simply denoted by t^-. The notion of substitution also extends naturally, on the new set of variables and terms. We will call *positive substitutions* the ones involving positive variables and positive terms only. Let ϑ be a positive substitution and $V \subseteq Var$. We define

$$\vartheta^{|V^-} = \{x/t^{|V^-} \mid x/t \in \vartheta, x \notin V\} \cup \{x^-/t^{|V^-} \mid x/t \in \vartheta, x \in V\}.$$

A substitution ϑ can be seen as composed by two parts: one part (denoted by ϑ^P) mapping positive variables, and one part (denoted by ϑ^N) mapping the negative ones. More formally:

$$\vartheta^P = \{x/t \mid x/t \in \vartheta, x \in Var\}$$

$$\vartheta^N = \{x^-/t \mid x^-/t \in \vartheta, x^- \in Var^-\}$$

ϑ^P represents the produced bindings, and ϑ^N represent the bindings that have to be received. The $^-$ annotation affects the application of a substitution ϑ to a term t in the following way

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

The application of a substitution to a formula is defined analogously.

The reason why the application of ϑ to x^- can result in $\vartheta(x)^-$ (instead of $\vartheta(x)$) is related to the peculiarity of the input-mode constraint of GHC. In fact, it does not apply to a specific variable, but to the argument of the atom. Therefore, when an annotated variable is bound to a term t, all the variables occurring in t get under the influence of the input-mode constraint, and therefore they have to inherit the annotation.

The new notion of application differs from the standard one in that $\{x \in Var \cup Var^- \mid x\vartheta \neq x\}$ (the set of variables bound by ϑ) is now a superset of $\{x \in Var \cup Var^- \mid \vartheta(x) \neq x\}$ (the set of variables mapped by ϑ into different terms). In fact, even if $\vartheta(x^-) = x^-$, $x^-\vartheta$ can be different from x^-, since in this case the definition $x^-\vartheta = \vartheta(x^-)$ applies.

Example 4.7 *Consider the term* $t = f(x, x^-, y, y^-)$, *and consider the substitution* $\vartheta = \{x/g(z), y/h(w), y^-/h(a)\}$. *We have:*

$$\vartheta^P = \{x/g(z), y/h(w)\}$$

$$\vartheta^N = \{y^-/h(a)\}$$

and

$$t\vartheta = f(g(z), g(z^-), h(w), h(a)).$$

□

The notions of domain and co-domain are also extended consequently, in order to deal with the additional cases in which a variable can be bound by a substitution. Namely, we define:

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

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

An other characterization of the domain and the co-domain is given by the following proposition.

Proposition 4.8 *Let* ϑ *be a substitution. Then*

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

$$\mathcal{C}(\vartheta) = \bigcup_{x \in \mathcal{D}(\vartheta)} \mathcal{V}(\vartheta(x)) \cup \bigcup_{x \in \mathcal{D}(\vartheta^P), x^- \notin \mathcal{D}(\vartheta^N)} \mathcal{V}(\vartheta(x)^-).$$

Proof Immediate, by definition. □

The notion of composition $\vartheta_1\vartheta_2$ of two substitutions, ϑ_1 and ϑ_2 is extended consequently as follows

$$\forall x \in Var \cup Var^- x(\vartheta_1\vartheta_2) = (x\vartheta_1)\vartheta_2.$$

It is easy to see that the standard properties of the composition still hold, namely

$$(\vartheta_1\vartheta_2)\vartheta_3 = \vartheta_1(\vartheta_2\vartheta_3)$$

$$\vartheta\epsilon = \epsilon\vartheta = \vartheta$$

where ϵ is the empty substitution, and

$$F(\vartheta_1\vartheta_2) = (F\vartheta_1)\vartheta_2.$$

The idempotent substitutions, namely the substitutions ϑ such that $\vartheta\vartheta = \vartheta$, are characterized by the following proposition.

Proposition 4.9 *A substitution* ϑ *is idempotent iff* $\mathcal{C}(\vartheta) \cap \mathcal{D}(\vartheta) = \emptyset$.

Proof

ϑ *is idempotent* *iff*
$(\forall x \in Var \cup Var^- : (x\vartheta)\vartheta = x\vartheta)$ *iff*
$(\forall x \in Var \cup Var^- : \forall y : (y \in \mathcal{V}(x\vartheta) \Rightarrow y \notin \mathcal{D}(\vartheta))$ *iff*
$(\forall y : (y \in \mathcal{C}(\vartheta) \Rightarrow y \notin \mathcal{D}(\vartheta))$ *iff*
$\mathcal{C}(\vartheta) \cap \mathcal{D}(\vartheta) = \emptyset$.

□

A substitution binds all the annotated version of the variables which are in the domain. It is reasonable, therefore, to extend the notion of unifier by adding the requirement to make equal also the annotated version of the terms to be unified (and not only the terms themselves). Namely, a substitution ϑ is a unifier for a set of sets of terms M iff

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

All the other notions (ordering on substitutions, *mgu*, parallel composition), are extended in the straightforward way. As before, $\mathcal{U}(M)$ and $mgu(M)$ will denote the sets of idempotent unifiers and idempotent mgu's of a set of sets of terms M.

Example 4.10 *1. Consider the program*

$$\{p(f(a)) \leftarrow |.\}$$

and consider the goal

$$\leftarrow p(x), x = f(a)$$

We annotate the variable x, in $p(x)$, in order to express the input-mode constraint. Then

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

and

$$\vartheta_2 = \{x/f(a)\} \in mgu(x = f(a), y = y)$$

We observe that $\vartheta_1 \leq \vartheta_2$. In fact,

$$\mathcal{U}(S(\vartheta_2)) = \{\vartheta_2, \ldots\} \subseteq \mathcal{U}(S(\vartheta_1)) = \{\vartheta_1, \vartheta_2, \ldots\}.$$

Therefore we have

$$mgu(S(\vartheta_1)) = \{\vartheta_1\},$$

and

$$\vartheta_1 \hat{\circ} \vartheta_2 = \{\vartheta_2\} = \{\{x/f(a)\}\}$$

2. Consider now the program

$$\{p(f(a)) \leftarrow |., q(f(a)) \leftarrow |.\},$$

and consider the goal

$$\leftarrow p(x), q(x).$$

We have

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

and then

$$\vartheta_1 \hat{\circ} \vartheta_1 = \{\vartheta_1\} = \{\{x^-/f(a)\}\}$$

If we look at the final result, we can see that the goal can be refuted, in the first case, by a suitable ordering on the execution of the atoms ($x = f(a)$ before $p(x)$). In fact the final result contains a substitution that does not map any annotated variable. This is not true in the second case, and therefore no refutation is possible. □

It is possible to prove that all the related properties (propositions 2.2 and 4.4, and corollaries 2.3 and 4.5) still hold. In fact, the unification algorithm can be extended, without modifying its *structure*, in order to deal with the new notion of application of a substitution to a term. We give an extended version of the unification algorithm, based on the one presented in [Ap], that works on finite sets of pairs. Given a finite set of finite sets of terms M, consider the (finite) set of pairs

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

The unifiers of a set of pairs $\{< t_1, u_1 >, \ldots, < t_n, u_n >\}$ are defined as the unifiers of the set $\{\{t_1, u_1\}, \ldots, \{t_n, u_n\}\}$. 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

$$\{< x_1, t_1 >, \ldots, < x_n, t_n >\}$$

where

1. $\forall i = 1, \ldots, n : x_i \in Var \cup Var^-$,
2. $\forall i, j = 1, \ldots, n : i \neq j \Rightarrow x_i \neq x_j$,
3. $\forall i = 1, \ldots, n : x_i \notin \mathcal{V}(t_1, \ldots, t_n)$,
4. $\forall i = 1, \ldots, n : x_i \in Var \text{ and } t_i \neq x_i^- \Rightarrow x_i^- \notin \mathcal{V}(x_1, \ldots, x_n, t_1, \ldots, t_n)$,

A solved (finite) set of pairs $P = \{< x_1, t_1 >, \ldots, < x_n, t_n >\}$ is always unifiable. An idempotent most general unifier of P can be obtained as follow. Consider the substitution γ_P associated to P, defined by

$$\gamma_P = \{x_1/t_1, \ldots, x_n/t_n\}.$$

In general, γ_P is not a unifier of P, since $x_i^- \gamma_P$ can be different from $t_i^- \gamma_P$. For the same reason, γ_P can be not idempotent. This is one of the main differences from the standard theory of unifiers (see [LMM]), and it is due basically to the new rule by which a substitution can bind a variable. Anyway, an idempotent unifier of P can be easily derived from γ_P. In fact, define

$$\delta_P = \gamma_P \gamma_P.$$

It turns out that δ_P is one of the idempotent most general unifiers of P, as shown by the following proposition.

Proposition 4.11 *Let P be a solved set of pairs. Then $\delta_P \in mgu(P)$.*

Proof Let $P = \{< x_1, t_1 >, \ldots, < x_n, t_n >\}$ be in solved form. Then, for any $x \in Var \cup Var^-$, we have three cases:

1. $x = x_i$, for a given $i\,(1 \leq i \leq n)$. In this case, $x\delta_P = x\gamma_P\gamma_P = t_i\gamma_P =$ (by the properties (3) and (4) of the solved form) $= t_i$.

2. $x = x_i^-$, for a given $i\,(1 \leq i \leq n)$. In this case $x\delta_P = x\gamma_P\gamma_P = t_i^-\gamma_P$.

3. $x \neq x_i, x \neq x_i^-$, for all $i = 1, \ldots, n$. In this case $x\delta_P = x\gamma_P\gamma_P = x\gamma_P = x$.

(idempotency) We have to show that for any $x \in Var \cup Var^-$, $x\delta_P\delta_P = x\delta_P$.

1. If $x = x_i$, for a given $i\,(1 \leq i \leq n)$, then $x\delta_P\delta_P = t_i\delta_P = x\gamma_P\gamma_P =$ (by the properties (3) and (4) of the solved form) $= t_i = x\delta_P$.

2. If $x = x_i^-$, for a given $i\,(1 \leq i \leq n)$, then $x\delta_P\delta_P = t_i^-\gamma_P\gamma_P\gamma_P =$ (by the property (3) of the solved form) $= t_i^-\gamma_P = x\delta_P$.

3. If $x \neq x_i, x \neq x_i^-$, for all $i = 1, \ldots, n$, then $x\delta_P\delta_P = x\delta_P (= x)$.

(unifier) For each $i = 1, \ldots n$, we have $x_i\delta_P = t_i =$ (by the properties (3) and (4) of the solved form) $= t_i\delta_P$. Moreover, $x_i^-\delta_P = t_i^-\gamma_P =$ (by the property (3) of the solved form) $= t_i^-\gamma_P\gamma_P = t_i^-\delta_P$.

(most general) Let σ be a unifier of P. We show that $\delta_P\sigma = \sigma$.

1. If $x = x_i$, for a given $i\,(1 \leq i \leq n)$, then $x\delta_P\sigma = t_i\sigma =$ (since σ is a unifier of P) $= x_i\sigma = x\sigma$.

2. If $x = x_i^-$, for a given $i\,(1 \leq i \leq n)$, then $x\delta_P\sigma = t_i^-\gamma_P\sigma =$ (since σ is a unifier of P) $= t_i^-\sigma =$ (since σ is a unifier of P) $= x_i^-\sigma = x\sigma$.

3. If $x \neq x_i, x \neq x_i^-$, for all $i = 1, \ldots, n$, then $x\delta_P\sigma = x\sigma$.

□

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

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

- *Let* $\rightarrow^*$ *be the reflexive-transitive closure of the relation* $\rightarrow$, *i.e.*

$$P \rightarrow^* P' \Leftrightarrow \exists P_1, \ldots, P_n : P = P_1, P' = P_n \text{ and } P_1 \rightarrow \ldots \rightarrow P_n$$

- *Let* P_{sol} *be the set of the* solved forms *of* P, *i.e.*

$$P_{sol} = \{P' \mid P \rightarrow^* P', \text{ and } P' \text{ is solved}\}$$

The set of substitutions determined by the algorithm is

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

□

The following proposition shows the correctness and the termination properties of the extended unification algorithm.

Proposition 4.13

1. **(finiteness)** *If* P *is a finite set of pairs, then the relation* $\rightarrow$ *on the set* $\{P' \mid P \rightarrow^* P'\}$ *is* finitely branching *and* noetherian *(i.e.* terminating*).*
2. **(solved form)** *If a finite set of pairs* P *is in* normal form *(i.e. there exist no* P' *such that* $P \rightarrow P'$*), then* P *is in solved form.*
3. **(soundness)** *The extended unification algorithm is* sound *with respect to the (finite) sets of pairs, i.e.*

$$\Delta(P) \subseteq mgu(P)$$

4. **(completeness)** *The extended unification algorithm is* complete *with respect to the finite sets of finite sets of terms, i.e.*

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

5. **(soundness and completeness of failure)** $P \rightarrow^*$ *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 finite-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. To this aim is sufficient to show 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 $P_1 \rightarrow P_2 \rightarrow \ldots P_n \rightarrow \ldots$. 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)** If P' is solved, then, by proposition 4.11, $\delta_{P'}$ 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). To this aim, 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 $\{x/u\}\vartheta = \vartheta$. Then 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 4.14 *Let M be a set of sets of terms. If ϑ is an idempotent most general unifier of M, then ϑ is* relevant *(see [Ap]). 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 4.13*(1)*, *(2)* and *(3)*, if M is unifiable, then there exists a set of pairs P such that:

   - $M_{pairs} \rightarrow^* P$

- 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 4.15 *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 [Ed] for an easy proof. □

Corollary 4.16 *If $\vartheta \in mgu(M)$, then*

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

Proof If $\vartheta, \vartheta' \in mgu(M)$, then $\vartheta \leq \vartheta'$ and $\vartheta' \leq \vartheta$, i.e. $\vartheta \sim \vartheta'$. By lemma 4.15, 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 $\exists\tau : \vartheta\tau = \sigma$. Then

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

i.e., $\vartheta' \leq \sigma$. □

We prove now the completeness of the algorithm. By corollary 4.16, if $\vartheta, \vartheta' \in mgu(M)$, then $\vartheta' = \vartheta\rho$ for an appropriate ρ. By lemma 4.14, 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:

$$\vartheta_1 = \{x_1/y_1, \ldots, x_n/y_n\},$$

$$\vartheta'_1 = \{y_1/x_1, \ldots, y_n/x_n\},$$

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

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 $M_{pairs} \rightarrow^* P$, and $< t_1, u_1 >, \ldots, < t_n, u_n > \in P$, then $M_{pairs} \rightarrow^* P' = P \cup \{< u_1, t_1 >, \ldots, < u_n, t_n >\}$. Now, let

$$P_1 = \{< t, u > | \, t/u \in \vartheta_1\} = \{< x_1, y_1 >, \ldots, < x_n, y_n >\},$$

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

$$P'_1 = \{< t, u > | \, t/u \in \vartheta'_1\} = \{< y_1, x_1 >, \ldots, < y_n, x_n >\},$$

$$P'_2 = \{< t, u > | \, t/u \in \vartheta'_2\},$$

and assume

$$M_{pairs} \rightarrow^* P_1 \cup P_2 = \{< x_1, y_1 >, \ldots, < x_n, y_n >\} \cup P_2.$$

Then

$$M_{pairs} \rightarrow^* \{< x_1, y_1 >, \ldots, < x_n, y_n >\} \cup \{< y_1, x_1 >, \ldots, < y_n, x_n >\} \cup P_2,$$

and, therefore, since $\{< x_1, y_1 >, \ldots, < x_n, y_n >\}\{y_1/x_1, \ldots, y_n/x_n\} = \{< x_1, x_1 > , \ldots, < x_n, x_n >\}$, that is eliminated by step *(3)*,

$$M_{pairs} \rightarrow^* \{< y_1, x_1 >, \ldots, < y_n, x_n >\} \cup P_2\{y_1/x_1, \ldots, y_n/x_n\} = \{< y_1, x_1 >, \ldots, < y_n, x_n >\} \cup P'_2.$$

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 $P \rightarrow fail$.

   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.

□

This result implies that the set of the idempotent most general unifiers of M is finite and can be computed in finite time by a deterministic simulation of the extended unification algorithm described in definition 4.12 (the non-determinism of the relation $\rightarrow$ can be simulated via a simple backtracking).

The following proposition is the extension to the annotated case of proposition 2.2 and can easily be proved on the unification algorithm given in definition 4.12.

Proposition 4.17 *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. $mgu(M_1 \cup M_2^- \cup M_3) = \vartheta_1 mgu(M_2\vartheta_1^- \cup M_3\vartheta_1)$.

□

Given a set of substitutions Θ, and a set of (positive) variables V, the function $\mathcal{E}_V(\Theta)$ (*filter of Θ on V*) eliminates from Θ all the substitutions whose domain contains a variable in V^-. More formally:

$$\mathcal{E}_V(\Theta) = \{\vartheta \in \Theta \mid \mathcal{D}(\vartheta) \cap V^- = \emptyset\}$$

We will write $\mathcal{E}_V(\vartheta)$ to denote $\mathcal{E}_V(\{\vartheta\})$, and $\mathcal{E}(\Theta)$ to denote $\mathcal{E}_{Var}(\Theta)$.

4.3 Sequences of substitutions.

In the intended semantics of concurrent logic languages, an atom A can be *verified* only if all the input-mode constraints are satisfied. As shown in section 4.2, the input-mode constraints are expressed in the substitutions by the presence of annotated variables. To consider only the final resulting substitution (associated to an atom) is however not sufficient. In fact, a *flat* representation of the result is not powerful enough to model the *effects* of the possible interleavings in the executions of the atoms in a goal. The main effect is that by the interleaving the atoms can provide each other the bindings necessary for going on in the respective computations. In a sense, we have to *register* the whole history of the execution of the atom, and therefore we have to adopt the notion of *sequences of substitutions*. From an operational point of view, they represent all the substitutions involved during the refutation. From a declarative point of view, they represent all the substitutions involved during the fixpoint construction.

Definition 4.18 *The finite sequences of substitutions, with typical element s, are defined by the following (abstract) syntax*

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

□

The operation of concatenation on sequences is considered to be associative, so parentheses are not needed and we can write $\vartheta_1.\vartheta_1.\cdots.\vartheta_n$. The role of the squared brackets, in definition 4.18, is to separate the *critical sections*. Their meaning will be clarified by the definition of the *interleaving operator* (on sequences). The $^-$ operation extends naturally on sequences of substitutions and $s^{|V^-}$ will denote the sequence obtained from s by replacing in each element of s every variable $x \in V$ by x^-. We introduce the following notations. If S, S' are sets of sequences, then

$$S.S' = \{s.s' \mid s \in S, s' \in S'\},$$

$$[S] = \{[s] \mid s \in S\}.$$

If s is a sequence, ϑ is a substitution, and $s = \vartheta'.s'$, then

$$\vartheta \,\hat{\circ}\, s = (\vartheta \,\hat{\circ}\, \vartheta').s',$$

$$\vartheta \,\hat{\circ}\, [s] = [(\vartheta \,\hat{\circ}\, \vartheta').s'],$$

if Θ is a set of substitution, then

$$\Theta \,\hat{\circ}\, s = \bigcup_{\vartheta \in \Theta} \vartheta \,\hat{\circ}\, s.$$

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 $\#(S) = k$.

The following definition introduces the notion of *result* of a sequence 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 bind negative variables.

Definition 4.19 *Let V be a set of (positive) variables, let s be a sequence and let S be a set of sequences. $\mathcal{R}_V(s)$ and $\mathcal{R}_V(S)$ (*result of s and S w.r.t. V*) are defined by mutual recursion as follows*

- $\mathcal{R}_V(\vartheta) = \mathcal{E}_V(\vartheta)$
- $\mathcal{R}_V([s]) = \mathcal{R}_V(s)$
- $\mathcal{R}_V(s_1.s_2) = \mathcal{R}_V(\mathcal{R}_V(s_1) \,\hat{\circ}\, s_2)$
- $\mathcal{R}_V(S) = \bigcup_{s \in S} \mathcal{R}_V(s)$.

Note that, thanks to the associative property, a sequence of the form $s_1.s_2$ can always be reduced to a sequence of the form $[s_1'].s_2'$ or $\vartheta.s_2'$. Therefore, if $\vartheta_1, \vartheta_2, \ldots, \vartheta_n$ are the substitutions occurring in s (in this order), then $\mathcal{R}_V(s) = \mathcal{R}_V(\ldots \mathcal{R}_V(\mathcal{E}_V(\vartheta_1) \,\hat{\circ}\, \vartheta_2), \ldots \,\hat{\circ}\, \vartheta_n)$.

□

Definition 4.20 *Let s_1, s_2 be sequences of substitutions, and let S_1, S_2 be sets of sequences of substitutions.*

1. $$\begin{aligned} s_1 \parallel s_2 = \quad & \{\vartheta.s \mid \exists s' : \vartheta.s' = s_1, s \in s' \parallel s_2\} \\ \cup \quad & \{\vartheta.s \mid \exists s' : \vartheta.s' = s_2, s \in s' \parallel s_1\} \\ \cup \quad & \{[s'].s \mid \exists s'' : [s'].s'' = s_1, s \in s'' \parallel s_2\} \\ \cup \quad & \{[s'].s \mid \exists s'' : [s'].s'' = s_2, s \in s'' \parallel s_1\} \end{aligned}$$
2. $$S_1 \parallel S_2 = \bigcup_{s_1 \in S_1, s_2 \in S_2} s_1 \parallel s_2.$$

□

Remark 4.21 *The collection of sets of sequences is a commutative semi-group with respect to the interleaving operation, with $\emptyset$ as the neutral element, i.e.*

1. $S \parallel \emptyset = \emptyset \parallel S = \emptyset$
2. $S_1 \parallel S_2 = S_2 \parallel S_1$
3. $(S_1 \parallel S_2) \parallel S_3 = S_1 \parallel (S_2 \parallel S_3)$.

Proof Standard. □

Thanks to the associativety of the interleaving operation, we can omit parentheses and simply write $s_1 \parallel s_2 \parallel s_3 \cdots$ (and $S_1 \parallel S_2 \parallel S_3 \cdots$).

5 Towards a declarative characterization of the GHC success set.

5.1 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 (see [Ap]), firstly introduced by van Emden and Kowalski [vEK]. We recall some basic notions. Given a program W, the *Herbrand base with variables* $\mathcal{B}_W$ associated to the program is the set of all the possible atoms that can be obtained by applying the predicates of W to the elements of *Term*. The set *Term* consists of terms built on *Var* and on constructors of W.

Definition 5.1 *An interpretation of W is a set of pairs of the form $< A, s >$, where A is an atom in $\mathcal{B}_W$ and s is a sequence of substitutions on Var and Term. I_W will denote the set of all the interpretations of W.* □

I_W 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 5.2 *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*

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

We give now the definition of our *immediate consequence* operator.

In the following, we use the short notation $\bar{s}$ to denote a sequence of sequences of substitutions $s_1, \ldots, s_n$. Moreover, if $\bar{s} = s_1, \ldots, s_n$ and $\bar{A} = A_1, \ldots, A_n$, $< \bar{A}, \bar{s} >$ stands for $< A_1, s_1 >, \ldots, < A_n, s_n >$, and $Int(\bar{s})$ stands for $s_1 \parallel \cdots \parallel s_n$.

Definition 5.3 *The mapping $T_W : I_W \rightarrow I_W$, associated to a program W, is defined as follows:*

$$\begin{aligned} T_W(I) = \quad & \{< A, s > | \quad \exists H \leftarrow \bar{G} | \bar{B} \in W_{\mathcal{V}(A)} : \\ & \exists \bar{s}', \bar{s}'' \text{ locally independent on } \bar{G}, \bar{B}, A : \\ & < \bar{G}, \bar{s}' >, < \bar{B}, \bar{s}'' > \in I \\ & s \in [mgu(A^-, H).Int(\bar{s}')].Int(\bar{s}'')\} \\ \cup \quad & \{< A, \vartheta > | \quad \exists H \in \{x = x\}_{\mathcal{V}(A)} : \vartheta \in mgu(A, H)\} \end{aligned}$$

□

Proposition 5.4 *T_W is continuous.*

Proof Standard. □

Corollary 5.5 *The least fixpoint $lfp(T_W)$ of T_W exists, and $lfp(T_W) = \bigcup_{n \geq 0} T_W^n(\emptyset)$ holds.*

Proof Standard. □

We define now the least fixpoint semantics associated to a program W.

Definition 5.6 *The least fixpoint semantics of a program W is the set*

$$\begin{aligned} \mathcal{F}(W) = \{< \bar{A}, \vartheta > | \quad & \exists \bar{s} \text{ locally independent on } \bar{A} : \\ & < \bar{A}, \bar{s} > \in lfp(T_W) \\ & \vartheta \in (\mathcal{R}(Int(\bar{s})))_{|\mathcal{V}(\bar{A})}\} \end{aligned}$$

□

5.2 Equivalence results.

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

Lemma 5.7 *Let A, H be atoms. Let μ be an idempotent positive substitution with no variables in common with H. Then*

$$\mu mgu((A\mu)^-, H) = \mu \circ mgu(A^-, H).$$

Proof It is a particular case of proposition 4.17*(2)*. □

Lemma 5.8 *Let μ be a positive substitution. Let $\vartheta_1, \ldots, \vartheta_n$ be idempotent substitutions. We have*

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

then

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

Proof This lemma is a simple extension of corollary 4.5*(3)*. □

Lemma 5.9 *Let W be a GHC 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*

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

$$\Leftrightarrow$$

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

and

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

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 $\vartheta_i' \in mgu(A_i\rho^-, H_i)$, 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, for each $s \in (s_1 \parallel \cdots \parallel s_n)$, by lemmata 5.7 and 5.8, we have

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

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 5.10 *Let W be a GHC program. Let $\bar{A}$ be a sequence of atoms, and let μ be a positive substitution. Then*

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

$$\Leftrightarrow$$

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

and

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

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 5.9, apart from the first element of each s'_i that is chosen in $mgu(A_i\mu^-, H_i)$. Then we have

1. *$\bar{s}'$ is locally independent on $\bar{A}\mu$*
2. $< \bar{A}\mu, \bar{s}' > \in T_W^k(\emptyset)$
3. $(\mu\mathcal{R}_{D(\mu)}(Int(\bar{s}')))_{|\mathcal{V}(A)\cup D(\mu)} =$
   (by lemmata 5.7 and 5.8)
   $(\mathcal{R}(Int(\mu\bar{s}')))_{|\mathcal{V}(A)\cup D(\mu)} =$
   (since only the ϑ'_i's $\in mgu(A_i\mu^-, H_i)$ have variables in common with μ)
   $(\mathcal{R}(\cdots \parallel (\mu mgu(A_i\mu^-, H_i)) \parallel \cdots \parallel Int(\bar{s}'')))_{|\mathcal{V}(A)\cup D(\mu)} =$
   $(\mathcal{R}(\cdots \parallel ((\rho\rho^{-1}\mu) \,\hat{\circ}\, mgu(A_i\rho^-, H_i)) \parallel \cdots \parallel Int(\bar{s}'')))_{|\mathcal{V}(A)\cup D(\mu)} =$
   (since the s''_i's have no variables in common with ρ)
   $(\mathcal{R}(\cdots \parallel ((\rho\rho^{-1}\mu) \,\hat{\circ}\, mgu(A_i\rho^-, H_i)) \parallel \cdots \parallel Int(\rho\bar{s}'')))_{|\mathcal{V}(A)\cup D(\mu)} =$
   (by lemmata 5.7 and 5.8)
   $(\rho(\mathcal{R}(\cdots \parallel ((\rho^{-1}\mu) \,\hat{\circ}\, mgu(A_i\rho^-, H_i)) \parallel \cdots \parallel Int(\bar{s}''))))_{|\mathcal{V}(A)\cup D(\mu)} =$
   (by lemma 5.9)
   $(\mathcal{R}(\cdots \parallel (\rho.(\rho^{-1}\mu).mgu(A_i\rho^-, H_i)) \parallel \cdots \parallel Int(\bar{s}'')))_{|\mathcal{V}(A)\cup D(\mu)} =$
   (since $\rho_{-1}\mu = \rho_{-1} \,\hat{\circ}\, \mu$)
   $(\mathcal{R}(\cdots \parallel (\rho.\rho^{-1}.\mu.mgu(A_i\rho^-, H_i)) \parallel \cdots \parallel Int(\bar{s}'')))_{|\mathcal{V}(A)\cup D(\mu)} =$
   $(\mathcal{R}(\cdots \parallel (\mu.mgu(A_i\rho^-, H_i)) \parallel \cdots \parallel Int(\bar{s}'')))_{|\mathcal{V}(A)\cup D(\mu)} =$
   $(\mathcal{R}(\mu.(Int(\bar{s}')))_{|\mathcal{V}(A)\cup D(\mu)}$.

($\Leftarrow$) Analogous.

□

We now prove the equivalence of the fixpoint semantics we have defined and the operational semantics of GHC given in section 3.

Theorem 5.11 (Soundness) *Let W be a GHC program, and let $\bar{A}$ be a sequence of atoms. If $\leftarrow \bar{A} \overset{\vartheta}{\longmapsto}{}^* \square$, then, for $\vartheta' = \vartheta_{|\mathcal{V}(\bar{A})}$, we have $< \bar{A}, \vartheta' > \in \mathcal{F}(W)$, i.e.*

$\exists \bar{s}$ locally independent on $\bar{A}$ such that $< \bar{A}, \bar{s} > \in lfp(T_W)$ and $\vartheta' \in (\mathcal{R}(Int(\bar{s})))_{|\mathcal{V}(\bar{A})}$

Proof By induction on the number of steps k performed in the refutations.

($k = 1$) Assume $\leftarrow \bar{A} \overset{\vartheta}{\longmapsto}{}^1 \square$. Then $\bar{A}$ is composed by only one atom, say A, and one of the following cases hold.

(A is an ordinary atom) In this case there exists in $W_{\mathcal{V}(A)}$ a clause of the form $H \leftarrow |.$ such that $\vartheta \in mgu(A, H)$ and $\vartheta_{|\mathcal{V}(A)} = \epsilon$. Then $< A, \sigma > \in T_W^1(\emptyset)$ holds, for each $\sigma \in mgu(A^-, H)$. Since $\vartheta_{|\mathcal{V}(A)} = \epsilon$, there exists $\sigma \in mgu(A^-, H)$ such that σ does not map variables in A, i.e. σ is positive. Therefore $\mathcal{R}(\sigma)_{|\mathcal{V}(A)} = \{\epsilon\}$.

(A is a unification atom) Assume that A is an equality atom, and, for some x not occurring in A, $\vartheta \in mgu(A, x = x)_{|\mathcal{V}(A)}$. Then $< A, \vartheta > \in T_W^1(\emptyset)$ holds and, since ϑ is positive, $\mathcal{R}(\vartheta)_{|\mathcal{V}(A)} = \vartheta_{|\mathcal{V}(A)}$.

($k > 1$) Assume $\leftarrow \bar{A} \stackrel{\sigma\mu}{\longmapsto}_{k_1} \leftarrow \bar{A}'\sigma\mu \stackrel{\vartheta'}{\longmapsto}_{k_2} \square$ and $\vartheta = (\sigma\mu\vartheta')_{|\mathcal{V}(\bar{A})}$. Let A_i be the atom in $\bar{A}$ selected for the first derivation step. We have two cases.

(A_i is an ordinary atom) In this case, there exists a clause $H \leftarrow \bar{G}|\bar{B}$ in $W_{\mathcal{V}(\bar{A})}$ such that:

- $\sigma \in mgu(A_i, H)$
- $\leftarrow \bar{G}\sigma \stackrel{\mu}{\longmapsto}_{k_1} \square$
- $k = k_1 + k_2 + 1$ (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

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

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

Then, by lemma 5.10 *(2)*, we have that there exists $\bar{s}'_1$ such that

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

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

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

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

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

By lemma 5.10 *(2)*, there exists $\bar{s}''_1$ such that

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

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

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

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

(A_i **is a unification atom**) This case can be reduced to the previous one, by considering H as $x = x$ and $\bar{G}$ and $\bar{B}$ as the empty sequences.

□

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

Theorem 5.12 **(Completeness)** *Let W be a GHC program, let $\bar{A}$ be a sequence of atoms and let $\bar{s}$ be a sequence of sequence of substitution locally independent on $\bar{A}$.*
If

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

$\vartheta \in (\mathcal{R}(Int(\bar{s})))_{|\mathcal{V}(\bar{A})}$,

then there exists ϑ' such that

$\leftarrow \bar{A} \stackrel{\vartheta'}{\longmapsto}{}^{*} \square$, and

$\vartheta = \vartheta'_{|\mathcal{V}(\bar{A})}$

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 $\vartheta = (\mathcal{E}(\vartheta'))_{|\mathcal{V}(\bar{A})} = \vartheta'_{|\mathcal{V}(\bar{A})}$. One of the following cases holds.

(A **is an ordinary atom**) In this case, there exists a clause of the form $H \leftarrow \mid \in W_{\mathcal{V}(A)}$ and $\vartheta' \in mgu(A^-, H)$ holds. Then, since $\mathcal{E}(\vartheta') \neq \emptyset$, we have $\vartheta'_{|\mathcal{V}(\bar{A})} = \epsilon$.

(A **is a unification atom**) In this case, there exists x such that x does not occur in A and $\vartheta' \in mgu(A^-, x = x)$.

In both cases, we have

$\leftarrow A \stackrel{\vartheta'}{\longmapsto}{}^{\perp} \square$.

($\#(\bar{s}) > 1$) 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, one of the following cases holds.

(A_i **is an ordinary atom**) There exists a clause with empty guard, $H \leftarrow |\bar{B} \in W_{\mathcal{V}(A_i)}$, such that $\sigma \in mgu(A_i^-, H)$. Moreover, since $\mathcal{R}(s) \neq \emptyset$, we have $\sigma_{|\mathcal{V}(A_i)} = \epsilon$. Therefore

$$\leftarrow \bar{A} \overset{\sigma}{\longmapsto}\!\!\downarrow \leftarrow (A_1, \ldots, A_{i-1}, \bar{B}, A_{i+1}, \ldots, A_n)\sigma$$

and

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

By lemma 5.10 *(1)*, there exist $\bar{s}', s'', \bar{s}''$ such that

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

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

(A_i **is a unification atom**) There exists x such that x does not occur in A_i and $\sigma \in mgu(A_i, x = x)$. This case can be reduced to the previous one, by considering H as $x = x$ and $\bar{G}$ and $\bar{r}$ as the empty sequences.

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

   From lemma 5.10 *(1)*, 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 inductive hypothesis, for each $r' \in Int(\bar{r}')$

   $$\leftarrow \bar{G}\sigma \overset{\tau}{\longmapsto}\!\!\overset{k}{\downarrow} \Box,$$

   (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

   $$\leftarrow \bar{A} \overset{\sigma\tau}{\longmapsto}\!\!\overset{k+1}{\downarrow} \leftarrow (A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n, \bar{B})\sigma\tau.$$

   The rest follows as in case (1).

□

6 Conclusions and future work.

The classical notions of substitution and unifier, that are the basis for the semantics theory of HCL, are not fully adequate to deal with the concurrent versions of HCL, such as GHC. The synchronization mechanisms of these languages, in fact, are based on constraints that restrict the standard notion of unificability. In this paper we have extended the theory of unification by introducing the concept of annotated variable and by considering the substitutions as composed by two parts. The annotated variables represent the variables affected by the input-mode constraints. The substitutions then contain two kinds of information: the bindings that are produced (bindings on not-annotated variables) and the bindings that have to be received (bindings on the annotated variables). The application of a substitution to a term presents an asymmetry in dealing with annotated and not-annotated variables. This asymmetry allows to model the unidirectionality of the communication flow represented by shared variables. Also the notion of unifier preserves this asymmetry, and the extended definition of most general unifier formalizes, in a declarative way, the constrained most general unification used in GHC.

Based on these concepts, we have extended the immediate consequence operator of HCL. The least fixpoint of such an operator represents the declarative counterpart of the success set of GHC, and allows to define a semantics for GHC in a declarative style. Finally we have showed the full equivalence of this declarative semantics, and the operational one, based on the notion of success set.

The method here described for the semantics of GHC should be easily applicable to other concurrent logic languages based on similar synchronization mechanisms, namely, Concurrent Prolog and PARLOG. The construction should remain essentially the same, since it is, in a sense, parametric with respect to the unification theory. The only requirement is that the properties on parallel composition (and composition) of substitutions are preserved, since they are used in the proofs of soundness and correctness. Therefore, a declarative semantics of PARLOG and Concurrent Prolog can be obtained by simply defining an adequate unification notion, where 'adequate' means that it has to model their synchronization mechanisms and to preserve the properties mentioned above.

This methodology strictly resembles the basic ideas of 'the Scheme' [JLM] and of CLP [JLM] and could be interesting to develop a general theory of the method we have used, so to obtain the counterpart of CLP for concurrent logic languages.

An other direction of work is the integration of the declarative characterization of the success set (affected by the synchronization mechanisms) with the declarative characterization of the failure set (affected by the commit operator).

Finally, we are thinking about a possible extension of the semantics described in this paper in order to obtain a declarative characterization of logic perpetual process.

7 Acknowledgements.

I would like to thank the people of the department of Software Technology, in the persons of Krzysztof R. Apt, Jaco W. de Bakker, and Jan-Willem Klop for providing a nice and stimulating environment to work in. In particular, I am grateful to Krzysztof R. Apt for many interesting discussions and for having invited me at the CWI. Discussions in the so-called Gaudi group (including Frank S. de Boer, Joost N. Kok and Jan J.M.M. Rutten) served as a starting point for this paper, which owes its existence to their stimuli, suggestions and encouragements. I acknowledge their great contribution to this work, not only from the scientific point of view, but also from the side of an incomparable human experience. Most of the results of this paper are an extension of ideas already presented in [BKPR].

References

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

[AvE] K.R. Apt, M.H. van Emden, *Contributions to the theory of logic programming,* JACM Vol. 29, No. 3, July 1982, pp. 841-862.

[Be] 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 225, Springer Verlag, 1986, pp. 335-349.

[BK] J.W. de Bakker and J.N. Kok, *Uniform Abstraction, Atomicity and Contractions in the Comparative Semantics of Concurrent Prolog,* Report CS-R88.., Centre for Mathematics and Computer Science, Amsterdam, 1988. Also in Proc. of FGCS'88.

[BKPR] F.S de Boer, Joost N. Kok, C. Palamidessi and J.J.M.M. Rutten *Control Flow versus Logic: a Denotational and a Declarative Model for Guarded Horn Clauses,* in preparation.

[Cl] K.L. Clark, *Predicate Logic as a Computational Formalism,* Res. Report DOC 79/59, Dept. of Computing, Imperial College, London, 1979.

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

[Ed] E. Eder, *Properties of Substitutions and Unifications,* J. Symbolic Computation 1, 1985, pp. 31-46.

[vEK] M.H. van Emden, R.A. Kowalski, *The Semantics of Predicate Logic as a Programming Language,* J. ACM, vol. 23, No. 4, 1976, pp. 733-742.

[FL] M. Falaschi, G. Levi, *Operational and fixpoint semantics of a class of committed-choice languages,* Proc. of the Int. Symposium on Logic Programming, IEEE Comp. Society Press, Seattle, 1988.

[FLMP] M. Falaschi, G. Levi, C. Palamidessi and M. Martelli *A more general Declarative Semantics for Logic Programming Languages.,* submited for publication on Theoretical Computer Science, 1988.

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

[JLM] J. Jaffar and J.-L. Lassez *Constraint Logic Programming* Proc. of the SIGACT-SIGPLAN Symp. on Principles of Programming Languages. ACM, 1987, pp. 111-119.

[JLM] J. Jaffar, J.-L. Lassez and M.J. Maher *A Logic Programming Language Scheme* in Logic Programming, Functions, Relations and Equations. (D. de Groot and G. Lindstrom eds.), Prentice Hall, 1986, pp. 441-468.

[K] J.N. Kok, *A compositional semantics for Concurrent Prolog,* Proc. Symp. on Theoretical Aspects Computer Science 1988 (R. Cori ed.), Lecture Notes in Computer Science 294, Springer Verlag, 1988, pp. 373-388.

[Le] G. Levi, *A new declarative semantics of Flat Guarded Horn Clauses,* Techn. Report, ICOT, Tokyo, 1988.

[LP1] G. Levi, C. Palamidessi, *The declarative semantics of logical read-only variables,* Proc. Symp. on Logic Programming, IEEE Comp. Society Press, 1985, pp. 128-137.

[LP2] G. Levi, C. Palamidessi, *An approach to the declarative semantics of synchronization in logic languages,* Proc. 4th Int. Conference on Logic Programming, Melbourne, 1987, pp. 877-893.

[Ll] J.W. Lloyd, *Foundations of Logic Programming,* Springer Verlag, 1984, (Second edition 1987).

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

[LS] J.-W. Lloyd and J.C. Shepherdson, *Partial Evaluation in Logic Programming,* Technical Report CS-87-09, Dept of Computer Science, University of Bristol, 1987.

[Sa] 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.

[Sa1] V.A. Saraswat: *GHC: Operational semantics, problems and relationship with CP(↓ , \|)* in: IEEE Int. Symp. on Logic Programming, San Francisco, 1987, pp. 347-358.

[Sh1] E.Y. Shapiro, *A subset of Concurrent Prolog and its interpreter,* Tech. Report TR-003, ICOT, Tokyo, 1983.

[Sh2] E.Y. Shapiro, *Concurrent Prolog, a progress report,* in Fundamentals of Artificial Intelligence (W. Bibel, Ph. Jorrand, eds.), Lecture Notes in Computer Science 232, Springer Verlag, 1987.

[Ue] K. Ueda, *Guarded Horn clauses,* PhD thesis, University of Tokyo, 1986.