数理解析研究所講究録
第618巻 1987年 40-58

40

Fixed Point Semantics for Parallel Logic Programming Languages

Etsuya Shibayama
(柴山悦哉)
Department of Information Science, Tokyo Institute of Technology,
(東京工業大学 情報科学科)
Oh-Okayama, Meguro-Ku,
Tokyo, Japan, 152

Abstract

A semantics description scheme is presented for parallel logic programming languages. This scheme consists of two parts: description of the semantics of unification and that of goal reduction strategies. For the former, we define a semantic domain each of whose elements naturally represents a variable binding environment, and, for the latter, we employ the power domain technique. As application examples of this scheme, we describe the semantics of Guarded Horn Clauses (GHC) and Concurrent Prolog (CP).

1 Introduction

One of the most significant features of logic programming languages is their semantic clearness. Especially, the semantics of pure-Prolog can be described in an elegant manner using a (complete) Herbrand base [Emden76], [Apt82], [Lloyd84].

However, most parallel logic programming languages employ mechanisms which are not based on *logic*. For instance, Parlog [Clark86], Concurrent Prolog (CP) [Shapiro83a], [Shapiro83b], Guarded Horn Clauses (GHC) [Ueda85], [Ueda86], and similar other languages employ a guard mechanism for the synchronization purpose. During execution of a program in such a language, a unification may be suspended until some variable(s) are bound to more specific term(s). This kind of mechanism is hard to be expressed in terms of Herbrand bases since they consist of just variable free terms.

In Section 2, we propose a new semantic domain each of whose elements represents some variable binding environment. This domain is constructed from not only variable free terms but also terms in which variables occur. The semantic domain introduced in Section 2 is powerful enough to represent an infinite sequence of unification processes based on either the Robinson's original unification [Robinson65] algorithm or its occur-check-free version.

The important problem is that the set of the (possible infinite) terms constituted by infinite number of symbols is not *compact* under the ordinary topology, and thus this space does not share several elegant properties of complete Herbrand bases.

In Section 3, in order to demonstrate the expressive power of this semantic domain, we describe the semantics of the guard mechanism which GHC employs. Also, with the power domain technique developed by [Clinger81], we construct a fixed point semantics of GHC. In Section 4, we describe the semantics of CP in the similar manner.

2 Semantic Domain of Variable Binding Environments

In this section, we define a semantic domain each of whose elements represents some variable binding environment. Each element of this domain is an equivalence relation on finite terms. We consider that a sequence of unifications defines an equivalence relation on terms. Also we show several significant features of this domain.

Definition 2.1

- *P is a finite set of horn clauses.*
- *F is the set of the functor symbols in P.*
- *For $f \in F$, $ar(f)$ is the arity of f.*
- *V is a countable set of variables.*
- *$Term_{F,V}$ is the set of the finite terms which are formed out of the symbols in F and V.* □

In order to avoid the situations where all the elements of F are constants, we assume that the binary functor . (dot) always belongs to F. In this paper, $t_1 \,.\, t_2 \,.\, \cdots \,.\, t_n$ is abbreviated as $\langle t_1, t_2, \cdots, t_n \rangle$.

Definition 2.2 *$E \subseteq 2^{Term_{F,V} \times Term_{F,V}}$ is the set of the binary relations each of which, say e, satisfies the following conditions.*

- *e is an equivalence relation.*
- *For all $t_1, ..., t_n, t'_1, ..., t'_n$ in $Term_{F,V}$ and f in F such that $ar(f) = n$,*

$$\{(t_i, t'_i)\}_{i=1}^{n} \subseteq e \text{ if and only if } (f(t_1, ..., t_n), f(t'_1, ..., t'_n)) \in e \quad \square$$

Definition 2.3 *Suppose that e and e' belong to E. A partial order relation $\sqsubseteq$ on E is defined as follows:*

$$e \sqsubseteq e' \overset{def}{\Longleftrightarrow} e \subseteq e' \quad \square$$

In the sequel, we show that the domain E (with $\sqsubseteq$) is a complete lattice and appropriate as a semantic domain of variable binding environments.

Lemma 2.4 *Let I be an index set of arbitrary cardinality. Suppose that e_i belongs to E $(\forall i \in I)$. There exists $\sqcap_{i\in I} e_i$, which is equal to $\cap_{i\in I} e_i$.*

Proof: *It is sufficient to prove $\cap_{i\in I} e_i \in E$, that is, to prove that $\cap_{i\in I} e_i$ satisfies the conditions in Definition 2.2.* □

Definition 2.5 *Suppose that r is a subset of $Term_{F,V} \times Term_{F,V}$.*

$$r^c \overset{def}{\Longleftrightarrow} \sqcap\{e \in E | r \subseteq e\} \quad \square$$

Obviously, r^c belongs to E. r^c is the least element of E among those which include r.

Theorem 2.6 $(E, \sqsubseteq)$ *is a complete lattice.*

Proof: *Suppose that I is an index set of arbitrary cardinality and that e_i belongs to E $(\forall i \in I)$. We prove that $(\cup_{i\in I} e_i)^c$ is the least upper bound of $\{e_i\}_{i\in I}$ in $(E, \sqsubseteq)$.*

By Definition 2.5, $(\cup_{i\in I} e_i)^c \supseteq \cup_{i\in I} e_i$. Therefore, $(\cup_{i\in I} e_i)^c$ is an upper bound of $\{e_i\}_{i\in I}$.

On the other hand, suppose that e' is an upper bound of $\{e_i\}_{i\in I}$ in E. $e' \in \{e \in E | \cup_{i\in I} e_i \subseteq e\}$ is satisfied and thus $e' \sqsupseteq \sqcap\{e \in E | \cup_{i\in I} e_i \subseteq e\} = (\cup_{i\in I} e_i)^c$.

Consequently, $(\cup_{i\in I} e_i)^c$ is the least upper bound of $\{e_i\}_{i\in I}$ in E. □

The following lemmas and theorem suggest that, for each $t_1, ..., t_n, t'_1, ..., t'_n \in Term_{F,V}$, after completing a sequence of unifications: $t_1 = t'_1$, $t_2 = t'_2$, ..., $t_n = t'_n$ successfully, the variable binding environment will be represented by $\{(t_1, t'_1), ..., (t_n, t'_n)\}^c$.

Lemma 2.7 *Let e be $\{(t_1, t'_1), ..., (t_n, t'_n)\}^c$. Suppose that $\langle t_1, \cdots, t_n\rangle$ and $\langle t'_1, \cdots, t'_n\rangle$ are unifiable and that the most general unifier of them is $\theta = \{s_1/v_1, ..., s_m/v_m\}$. It is satisfied that $\{(v_1, s_1), ..., (v_m, s_m)\} \subseteq e$.*

Proof: *This lemma can be proven by induction on the total steps necessary to unify $\langle t_1, \cdots, t_n\rangle$ and $\langle t'_1, \cdots, t'_n\rangle$ using the Robinson's unification algorithm.* □

Lemma 2.8 *Let r be a subset of $Term_{F,V} \times Term_{F,V}$.*

$$r^c = \{(t, t') | \exists n \text{ s.t. } t \overset{n}{\underset{r}{\Longrightarrow}} t'\}$$

is satisfied, where the relation $\overset{n}{\underset{r}{\Longrightarrow}}$ is defined as follows:

1. $t \overset{0}{\underset{r}{\Longrightarrow}} t$ $(t \in Term_{F,V})$.
2. $t \overset{0}{\underset{r}{\Longrightarrow}} t'$ *if* $(t, t') \in r$.

3. $t \overset{k+1}{\underset{r}{\Longrightarrow}} t'$ *if* $t' \overset{k}{\underset{r}{\Longrightarrow}} t$.

4. $t \overset{k+j+1}{\underset{r}{\Longrightarrow}} t''$ *if there exists* t' *such that* $t \overset{k}{\underset{r}{\Longrightarrow}} t'$ *and* $t' \overset{j}{\underset{r}{\Longrightarrow}} t''$.

5. $t_i \overset{k+1}{\underset{r}{\Longrightarrow}} t_i'$ $(1 \leq i \leq n)$ *if there exist two terms* $f(t_1,...,t_n)$ *and* $f(t_1',...,t_n')$ *such that* $f(t_1,...,t_n) \overset{k}{\underset{r}{\Longrightarrow}} f(t_1',...,t_n')$.

6. $f(t_1,...,t_n) \overset{1+k_1+...+k_n}{\underset{r}{\Longrightarrow}} f(t_1',...,t_n')$ *if* $t_i \overset{k_i}{\underset{r}{\Longrightarrow}} t_i'$ $(1 \leq i \leq n)$.

Proof: *Let* r^* *be the set* $\{(t,t')|\exists n \text{ s.t. } t \overset{n}{\underset{r}{\Longrightarrow}} t'\}$. *Obviously,* r^* *is an upper bound of* r *and belongs to* E. *This means that* $r^* \sqsupseteq r^c$. *On the other hand,* $r^c \supseteq \{(t,t')|t \overset{n}{\underset{r}{\Longrightarrow}} t\}$ *can be proven by induction on* n. *Therefore,* $r^* = r^c$. □

Theorem 2.9 *Let* e *be* $\{(t_1,t_1'),...,(t_n,t_n')\}^c$. *Suppose that* $\langle t_1,\cdots,t_n\rangle$ *and* $\langle t_1',\cdots,t_n'\rangle$ *are unifiable and their most general unifier is* θ. *For all* t *and* t' *which belong to* $Term_{F,V}$, *the following is satisfied.*

$$(t,t') \in e \quad \text{if and only if} \quad t\theta = t'\theta$$

Proof: *Let* r *be* $\{(t_1,t_1'),...,(t_n,t_n')\}$. *For all* k, *the following can be proven by induction on* k.

$$t \overset{k}{\underset{r}{\Longrightarrow}} t' \quad \text{implies} \quad t\theta = t'\theta$$

Thus, if $(t,t') \in e = r^c$ *is satisfied,* $t\theta = t'\theta$ *is also satisfied. On the other hand, if* $t\theta = t'\theta$ *is satisfied, they must have the most general unifier* $\sigma = \{s_1/v_1,...,s_m/v_m\}$ *such that* $t\sigma = t'\sigma$ *and* $\{(s_1,v_1),...,(s_m,v_m)\} \subseteq e$. *In this case,* $(t,t') \in e$ *can be proven by induction on the height of* t *and/or* t'. □

Definition 2.10 *Suppose* $e \in E$ *and* $t \in Term_{F,V}$.

1. $[t]_{e|V} \overset{def}{\equiv} \{t'|(t,t') \in e \text{ and } \exists\theta,\theta^{-1} \text{ s.t. } t\theta = t', t'\theta^{-1} = t\}$

2. $[t]_e \overset{def}{\equiv} \{[t']_{e|V}|(t,t') \in e\}$

3. $[t_1]_{e|V} \sqsubseteq [t_2]_{e|V} \overset{def}{\Longleftrightarrow} \exists\theta \text{ s.t. } t_1\theta = t_2$ □

Obviously, the elements in $[t]_{e|V}$ (for each t) are invariants. Notice that the relation $\sqsubseteq$ defined above is just a pre-order in $\{[t]_{e|V}|t \in Term_{F,V}\}$.

Lemma 2.11 *The relation* $\sqsubseteq$ *defined in Definition 2.10 is a partial order relation on each* $[t]_e(t \in Term_{F,V})$.

Proof: *First, we show that $\sqsubseteq$ is a well-defined relation. Assume that $t'_1 \in [t_1]_{e|V}$ and $t'_2 \in [t_2]_{e|V}$. In this case, by the definition, there exist θ_1 and θ_2 such that $t_1\theta_1 = t'_1$, $t'_1\theta_1^{-1} = t_1$, $t_2\theta_2 = t'_2$, and $t'_2\theta_2^{-1} = t_2$. If $t_1\theta = t_2$ is satisfied for some θ, $t'_1\theta_1^{-1}\theta = t'_2\theta_2^{-1}$ and thus $t'_1\theta_1^{-1}\theta\theta_2 = t'_2$ is also satisfied. Therefore, $\sqsubseteq$ is well-defined.*

Next, we prove that $\sqsubseteq$ is a partial ordered relation on $[t]_e$. Suppose that $[t']_{e|V}$, $[t_1]_{e|V}$, $[t_2]_{e|V}$, and $[t_3]_{e|V}$ are arbitrary elements of $[t]_e$.

- *For a null substitution λ, $t'\lambda = t'$ is satisfied. Therefore, $[t']_{e|V} \sqsubseteq [t']_{e|V}$.*
- *Suppose that both $[t_1]_{e|V} \sqsubseteq [t_2]_{e|V}$ and $[t_2]_{e|V} \sqsubseteq [t_3]_{e|V}$ are satisfied. There exist θ_1 and θ_2 such that $t_1\theta_1 = t_2$ and $t_2\theta_2 = t_3$. In this case, $[t_1]_{e|V} \sqsubseteq [t_3]_{e|V}$ is satisfied since $t_1\theta_1\theta_2 = t_3$.*
- *Suppose that both $[t_1]_{e|V} \sqsubseteq [t_2]_{e|V}$ and $[t_2]_{e|V} \sqsubseteq [t_1]_{e|V}$ are satisfied. There exist θ_1 and θ_2 such that $t_1\theta_1 = t_2$ and $t_2\theta_2 = t_1$. In this case, $[t_1]_{e|V} = [t_2]_{e|V}$ by Definition 2.10.* □

Example: Suppose that $e = \{(X, f(Y)), (Y, Z)\}^c$ where f is a functor and X, Y, and Z are variables. If g is a binary functor:

$$[g(f(Y), f(Y))]_{e|V} = [g(f(Z), f(Z))]_{e|V} \supsetneqq [g(f(Y), f(Z)]_{e|V} \supsetneqq [g(X, f(Z))]_{e|V}.$$

The following theorem shows that it can be described in terms of the semantic domain E whether or not given two terms are unifiable.

Theorem 2.12 *Let e be $\{(t_1, t'_1), ..., (t_n, t'_n)\}^c$. $\langle t_1, \cdots, t_n\rangle$ and $\langle t'_1, \cdots, t'_n\rangle$ are unifiable if and only if, for all $t \in Term_{F,V}$, the greatest element in $[t]_e$ exists.*

Proof: *Suppose first that $\langle t_1, \cdots, t_n\rangle$ and $\langle t'_1, \cdots, t'_n\rangle$ are unifiable and their most general unifier is θ. By Theorem 2.9, for each element $[t']_{e|V}$ in $[t]_e$, $t'\theta = t\theta$ is satisfied. Therefore, $[t']_{e|V} \sqsubseteq [t\theta]_{e|V} \in [t]_e$ for all t' s.t. $[t']_{e|V} \in [t]_e$. This implies that $[t\theta]_{e|V}$ is the greatest element in $[t]_e$.*

Next, suppose that $\langle t_1, \cdots, t_n\rangle$ and $\langle t'_1, \cdots, t'_n\rangle$ are not unifiable. There are two cases. That is, during application of the Robinson's unification algorithm, either:

1. *some variable v and a term $t'(v)$ in which v occurs, or*
2. *two terms t' and t'' whose principal functors are different*

are attempted to be unified. In the former case, $(v, t'(v)) \in e$ can be proven similarly to Theorem 2.9. Therefore, $[v]_{e|V}, [t'(v)]_{e|V}, [t'(t'(v))]_{e|V}, ...$ belong to the same component $[v]_e$. As a result, $[v]_e$ does not have its greatest element. In the latter case, it can be proven that $(t', t'') \in e$. Therefore, $[t']_{e|V}$ and $[t'']_{e|V}$ belong to the same component $[t']_e = [t'']_e$. However, no element in the component can be greater than or equal to both $[t']_{e|V}$ and $[t'']_{e|V}$. Consequently, $[t']_e = [t'']_e$ does not have its greatest element. □

Example: Let f and g be unary and binary functors, respectively. Also let X, Y, and Z be variables.

1. Suppose that $e = \{(X, f(Y)), (Y, Z)\}^c$.
   The greatest element of $[g(X,X)]_e$ is $[g(f(Y), f(Y))]_{e|V} = [g(f(Z), f(Z))]_{e|V}$.
2. Suppose that $e = \{X, f(X)\}^c$.
   $[X]_e$ contains infinitely many elements $[X]_{e|V}$, $[f(X)]_{e|V}$, $[f(f(X))]_{e|V}$, ...,$[f(f(f(...(X)...)))]_{e|V}$, ... and there is no greatest element.
3. Suppose that $e = \{f(X), g(Y,Z)\}^c$.
   $[f(X)]_e$ contains $[f(X)]_{e|V}$ and $[g(Y,Z)]_{e|V}$ and there is no element that is greater than both of them. □

By Theorem 2.9 and 2.12, it is shown that the semantics of a finite sequence of unifications can be described in terms of the semantic domain E. In order to describe the semantics of a possibly infinite sequence of unifications, we have to introduce infinite terms. For this purpose, first, we consider the completion of $Term_{F,V}$.

Definition 2.13

1. *$Term^*_{F,V}$ is the set of the possibly infinite terms which are formed out of the symbols in F and V.*
2. *For all $t, t' \in Term^*_{F,V}$, $d(t,t')$ is defined as follows:*
   (a) if $t = t'$, $d(t,t') = 0$
   (b) if $t \neq t'$, $d(t,t') = 2^{-\inf\{n | t \text{ and } t' \text{ differ at depth } n\}}$

   □

Note that the function d defined in Definition 2.13 satisfies the axioms of metric. However, $Term^*_{F,V}$ is not a compact space under this metric since there are countably many variables in V and, for each pair of them v and v' ($v \neq v'$), $d(v,v') = 1$. We consider the depth of a variable/constant is 0

In the sequel, we allow substitutions to substitute countably many variables.

Definition 2.14 *Let θ be $\{s_1/v_1, ..., s_n/v_n, ...\}$ and θ_n be $\{s_1/v_1, ..., s_n/v_n\}$ ($n \in \mathbf{N}$).*

$$t\theta \stackrel{def}{\equiv} \lim_{n\to\infty} t\theta_n \quad \square$$

Of course, such limits in $Term^*_{F,V}$ are defined according to the metric d.

Definition 2.15

1. *Suppose that $e \in E, t \in Term^*_{F,V}$.*

$$[t]^*_{e|V} = \left\{ t' \;\middle|\; \begin{array}{l} \exists\theta = \{v'_i/v_i\}_{i\in I} \text{ and } \theta^{-1} = \{v'_i/v_i\}_{i\in I} \text{ such that} \\ I \subseteq \mathbf{N}, \{(v_i, v'_i)\}_{i\in I} \subseteq e, t\theta = t' \text{ and } t'\theta^{-1} = t \end{array} \right\}$$

2. *Suppose that* $t \in Term_{F,V}$.

$$[t]^*_e = \{[t']^*_{e|V} | (t,t') \in e\} \cup \{[t']^*_{e|V} | \exists \{(t,t_i)\}_{i \in \mathbf{N}} \subseteq e \text{ s.t. } \lim_{i \to \infty} t_i = t'\}$$

3. *Let* $t, t' \in Term^*_{F,V}$.

$$[t]^*_{e|V} \sqsubseteq [t']^*_{e|V} \stackrel{def}{\Longleftrightarrow} \exists \theta \text{ s.t. } t\theta = t' \quad \square$$

Note that, for each $t \in Term_{F,V}$, $[t]^*_{e|V} = [t]_{e|V}$. Note also that we do not define $[t]^*_e$ for an infinite term t since we are only interested in such components of the terms which occur in a program or an initial goal. We implicitly assume that each program does not contain any infinite term.

Also, in this case, $\sqsubseteq$ is just a pre-order on $\{[t]^*_{e|V} | t \in Term^*_{F,V}\}$

Definition 2.16

$$d_e([t_1]^*_{e|V}, [t_2]^*_{e|V}) \stackrel{def}{=} min\{d(t'_1, t'_2) | t'_1 \in [t_1]^*_{e|V}, t'_2 \in [t_2]^*_{e|V}\} \quad \square$$

The function d_e also satisfies the axioms of metric. In the sequel, we regard the set $\{[t]^*_{e|V} | t \in Term^*_{F,V}\}$ as a metric space.

Lemma 2.17 *Let* $t \in Term_{F,V}$ *and* $\{[t_i]^*_{e|V}\}_{i \in \mathbf{N}} \subseteq [t]^*_e$. *Suppose that*

$$[t_1]^*_{e|V} \sqsubseteq [t_2]^*_{e|V} \sqsubseteq \ldots \sqsubseteq [t_i]^*_{e|V} \sqsubseteq \ldots$$

The limit element $\sqcup_{i=1}^{\infty} [t_i]^*_{e|V}$ *exists in* $[t]^*_e$.

Proof: *We first define the truncation of* t' $(\in Term^*_{F,V})$ *at depth* d, *say* $trunc(t', d)$, *as follows:*

1. *If* t' *is a variable or a constant,* $trunc(t', d) = t'$.
2. *Otherwise, suppose that* $t' = f(t'_1, \ldots, t'_n)$.
   (a) $trunc(t', 0) = f(\Omega, \ldots, \Omega)$ *where* Ω *is a special symbol which is not an elment of* $F \cup V$.
   (b) $trunc(t', d+1) = f(trunc(t'_1, d), \ldots, trunc(t'_n, d))$.

Furthermore, we define the truncation of $[t']^*_{e|V}$ *at depth* d, *say* $trunc([t']^*_{e|V}, d)$, *to be* $[trunc(t', d)]^*_{e|V}$ [1]. *We can easily prove that, for all* d, $\{trunc([t']^*_{e|V}, d) \mid [t']^*_{e|V} \in [t]^*_e\}$ *is a finite set by induction on* d. *This implies* $\forall d\ \exists i$ *s.t.* $trunc([t_i]^*_{e|V}, d) = trunc([t_{i+1}]^*_{e|V}, d) = trunc([t_{i+2}]^*_{e|V}, d) = \ldots$. *That is,* $d_e([t_i]^*_{e|V}, [t_{i+j}]^*_{e|V}) \leq 2^{-d}$ *(for all* j*). Therefore, by Definition 2.15,* $\lim_{i \to \infty} [t_i]^*_{e|V}$ *exists and belongs to* $[t]^*_{e|V}$ *It is easy to prove* $\lim_{n \to \infty} [t_i]^*_{e|V} = \sqcup_{i=1}^{\infty} [t_i]^*_{e|V}$
$\square$

[1] In this situation, the set of the functor symbols is not F but $F \cup \{\Omega\}$. Thus, precisely speaking, we must say $[trunc(t', d)]^*_{(e \cup \{(\Omega,\Omega)\})^c|V}$. However, for simplicity, we consider that e means $(e \cup \{(\Omega,\Omega)\})^c$ in such situations.

Definition 2.18 *Let* $t_i, t_i' \in Term_{F,V} (i \in \mathbf{N})$. *Two infinite lists* $\langle t_1, t_2, \cdots, t_i, \cdots \rangle$ *and* $\langle t_1', t_2', \cdots t_i', \cdots \rangle$ *are unifiable if and only if, for all* $n \in \mathbf{N}$, $\langle t_1, t_2, \cdots, t_n \rangle$ *and* $\langle t_1', t_2', \cdots, t_n' \rangle$ *are unifiable.* □

Theorem 2.19 *Let* e *be* $\{(t_1, t_1'), ..., (t_n, t_n'), ...\}^c$ $(t_i, t_i' \in Term_{F,V}, i \in \mathbf{N})$. *Suppose two infinite lists* $\langle t_1, t_2, \cdots, t_n, \cdots \rangle$ *and* $\langle t_1', t_2', \cdots, t_n', \cdots \rangle$ *are unifiable. For every* $t \in Term_{F,V}$, *there exists the greatest element in* $[t]_e^*$.

Proof: *Let* e_n *be* $\{(t_1, t_1'), ..., (t_n, t_n')\}^c$. *For all* n, $[t]_{e_n}$ *contains its greatest element, say* $[t^n]_{e_n|V}$. *Obviously,* $[t^1]_{e|V}^* \sqsubseteq [t^2]_{e|V}^* \sqsubseteq ... \sqsubseteq [t^n]_{e|V}^* \sqsubseteq ....$ *Therefore, their limit point* $\lim_{n\to\infty} [t^n]_{e|V}^* = \sqcup_{n=1}^{\infty} [t^n]_{e|V}^*$ *exists in* $[t]_e^*$. *Since* $\sqcup_{n=1}^{\infty} e_n = \cup_{n=1}^{\infty} e_n$, *for all* $t' \in Term_{F,V}^*$, *if* $[t']_{e|V}^* \in [t]_e^*$, *there exists* n *such that* $[t']_{e_n|V}^* \in [t]_{e_n}^*$ *and so* $[t']_{e|V}^* \sqsubseteq [t^n]_{e|V}^*$. *Consequently,* $\lim_{n\to\infty} [t^n]_{e|V}^*$ *is the greatest element in* $[t]_e^*$. □

In this case, even if two terms t and t' which cannot be unified, $[t'']_{\{(t,t')\}^c}$ $(\forall t \in Term_{F,V})$ may have the greatest element in it.

Example: $f(X)$ and X cannot be unified but, for every t in $Term_{F,V}$, $[t]_{\{(f(X),X)\}^c}^*$ contains its greatest element. For instance, the greatest element of $[X]_{\{(f(X),X)\}^c}^*$ is

$$[\overbrace{f(f(f\cdots))}^{\omega}]_{\{(f(X),X)\}^c|V} \quad \square$$

Usually, practical implementations of Prolog lack the occur check mechanism for the efficiency. Sometimes they are useful since we can construct infinite terms, called rational trees [Colmerauer82], by using the occur-check-free unification mechanism.

Suppose that $t(v)$ is a term in which a variable v occurs. Essentially, $\{(v, t(v))\}^c$ and $\{(v, t(v_1)), ..., (v_n, t(v_{n+1})), ...\}^c$ are equivalent in the sense that, for each term t' in which any of the variables $v_1, ..., v_n, ...$ does not occur, the greatest element in $[t']_{\{(v,t(v))\}^c}$ is equal to that of $[t']_{\{(v,t(v_1)),...,(v_n,t(v_{n+1})),...\}^c}$. This implies that, even if two terms t and t' are not unifiable by the original unification algorithm, $[t'']_{\{(t,t')\}^c}$ $(\forall t \in Term_{F,V})$ has the greatest element in it under the condition that they are unifiable by an occur-check-free unification algorithm.

In contrast, if two terms t and t' are not unifiable even by an occur-check-free unification algorithm, it is easily proven that $[t]_{\{(t,t')\}^c}^* (= [t']_{\{(t,t')\}^c}^*)$ cannot have its greatest element.

As a result, in this section, we have developed a denotational formalism of unifications. By using the latter half of the formalism, we can naturally express an infinite sequence of unifications. According to whether the former or the latter half of the formalism is used as failure detection, we can express either the Robinson's original algorithm or its occur-check-free version.

Definition 2.20 *Let* e *be an element of* E. *If, for every* t $(\in Term_{F,V})$, *the greatest element in* $[t]_e^*$ *exists,* e *is called a successful environment. Otherwise it is called a failure environment.* □

The formalism in this section has an advantage on representing a distributed binding environment faithfully. For instance, suppose that each of n individual processors has its own binding environment and that the global environment is defined as their union. If the binding environment of the i-th processor is represented as e_i $(1 \leq i \leq n)$, the global environment is represented as $\sqcup_{i=1}^{n} e_i$.

3 A Fixed Point Semantics of GHC

In this section, we describe the semantics of the goal reduction strategy which GHC employs. The most significant problem is how to describe the guard mechanism. We show that it can be described in terms of the semantic domain E. In the rest of the description, we use the power domain technique, which is based on the work of [Clinger81].

Definition 3.1 *A GHC program is a finite set of guarded program clauses.* □

The operational semantics of GHC can be found in [Ueda86].

Definition 3.2 *For a node N of some tree, we define as follows:*

- *$depth(N)$ = the depth of N. We assume that the depth of the root is 0.*
- *$parent(N)$ is the parent of N if it is not the root.*
- *$child(N,i)$ is the i-th child of N if it exists. Otherwise, $child(N,i)$ is undefined.*

A labeled tree is a tree each of whose nodes is attached to a label. If N is a node of a labeled tree T:

- *$l(N)$ is the label attached to the node N.*
- *$remove(T,N)$ is the labeled tree which is obtained by removing the subtree whose root is N from T.*
- *If N is a leaf of T, $add(T,N,l_1,...,l_n)$ is the labeled tree which is obtained by adding n children to the node N of T and attaching the labels $l_1,...,l_n$ to the children one by one in this order.*
- *$replace(T,N,N_1,...,N_n)$ is the labeled tree obtained by removing the subtree whose root is N from T and then letting the nodes $N_1,...,N_n$ be children of $parent(N)$ of T.*

If $N_1,...,N_n$ are distinct nodes of a labeled tree T, $T[N_1/l_1,...,N_n/l_n]$ is the labeled tree which is obtained by replacing $l(N_k)$ of T by l_n $(1 \leq k \leq n)$. □

Execution status of a GHC program is represented as an element of E introduced in the previous section [2] and a labeled tree. The former, called an environment, represents the

[2]Since GHC is a single environment language, a single element of E is sufficient to represent the current binding environment. However, when we describe the semantics of multiple environment languages such as CP, more than one element of E is necessary (See Section 4).

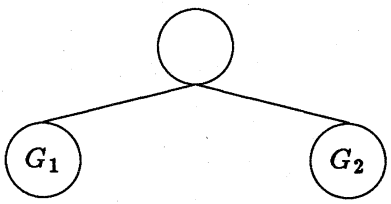

Figure 1: The Initial Execution Tree

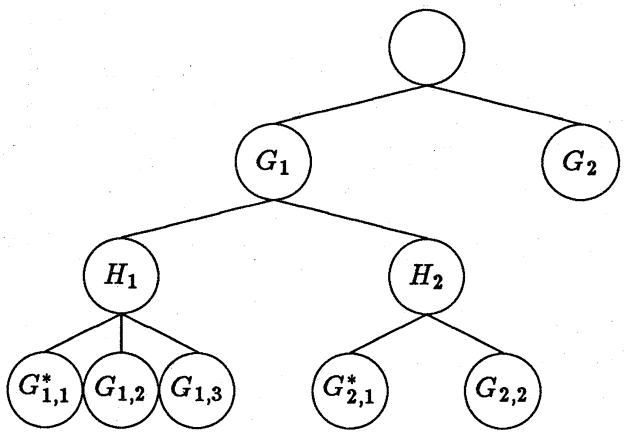

Figure 2: The Second Stage

current binding environment, whereas the latter, called an execution tree, represents how far the computation proceeds and which goals remain.

First, we illustrate the representation scheme of execution trees by an example. Assume that the initial goals are G_1, G_2. The initial execution tree is the labeled tree in Figure 1. In general, nothing (or, if necessary, a dummy symbol) is attached to the root node. At the initial stage, the environment is $\perp_E = \{(t,t)|t \in Term_{F,V}\}$.

Next, we assume that the goal G_1 has two candidate clauses:

$$H_1 :- G_{1,1}|G_{1,2}, G_{1,3}.$$
$$H_2 :- G_{2,1}|G_{2,2}.$$

and they are invoked by G_1. In this case, the execution tree becomes as in Figure 2. Precisely speaking, the literals except G_1 and G_2 are variants of those in the above clauses. From now on, we implicitly assume the existence of a systematic variable renaming strategy for avoiding name conflicts.

According to this representation scheme, a literal of GHC is attached to each non-root node. Each literal attached to a node of an odd depth represents a goal, whereas each literal attached to a node of an even depth represents the head of some candidate clause for the parent goal. Of course, on the nodes to which $G_{1,1}$ and $G_{2,1}$ are attached, extra information (illustrated $*$ in Figure 2) is stored which specifies these goals are in guard parts.

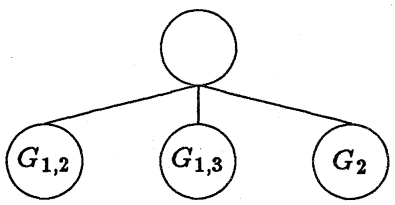

Figure 3: The Third Stage

Definition 3.3 *When a node contains the extra information mentioned above, we call it a guard node. In contrast, if a node to which a goal is attached does not contain the information, we call it a body node.* □

Simultaneously, by unifying G_1 and $H_i(i = 1,2)$, the environment is updated. For instance, supposing that $G_1 = p(a(X))$, $H_1 = p(Y)$, and $H_2 = p(a(Z))$, the environment becomes $\{(a(X), Y), (a(X), a(Z))\}^c$.

Next, if $G_{1,1}$ immediately succeeds and the clause $H_1 :- G_{1,1}|G_{1,2}, G_{1,3}.$ is committed, the execution tree is transformed into as in Figure 3. That is, the nodes which represent the clause $H_2 :- G_{2,1}|G_{2,2}.$ is discarded and G_1 is replaced by $G_{2,1}$ and $G_{2,3}$.

Definition 3.4 *An execution tree is a labeled tree which satisfies the following conditions:*

1. *If $depth(N)$ is odd, $l(N)$ is a goal in either some clause or the initial goal sequence. Otherwise, if N is not the root, $l(N)$ is a head of some clause.*
2. *Supposing that $l(parent(N))$ is a goal literal, $l(N)$ must be the head of some clause invoked by the goal. Otherwise, that is, $l(parent(N))$ is a head literal, $l(N)$ must be a goal invoked during reduction of the clause whose head is $l(parent(N))$. We assume that N contains extra information if $l(N)$ is invoked during reduction of the guard part of the clause.* □

For simplicity, in the sequel, we assume that each argument of a head is a distinct variable. Notice that each clause can be transformed into the form satisfying this assumption. For instance, a clause:

$$p(t_1, ..., t_m) :- G_1, ..., G_i|B_1, ..., B_j.$$

can be transformed into

$$p(v_1, ..., v_n) :- v_1 = t_1, ..., v_n = t_n, G_1, ..., G_i|B_1, ..., B_j.$$

where $v_1, ..., v_m$ are variables which do not occur in the former clause. Also we assume that GHC supports just one built-in predicate $=$.

The following definition describes the semantics of the guard mechanism which GHC employs in terms of the domain E.

Definition 3.5 *Let N be a leaf of some execution tree. If $l(N)$ is a goal $t = t'$, it is determined whether or not the goal is suspended under a successful environment e ($\in E$) by the following rule: If parent(N) is the root, it is not suspended. Otherwise, assume that $e' = e \sqcup \{(t,t')\}^c$:*

- *When N is a guard node, suppose that $l(parent(N))$ is $p(v_1,...,v_n)$ and that the greatest element of $[\langle v_1,\cdots,v_n\rangle]^*_e$ is $[t'']^*_{e|V}$. If $[t'']^*_{e'|V}$ is not the greatest element of $[\langle v_1,\cdots,v_n\rangle]^*_{e'}$, the goal is suspended. Otherwise, it is not suspended.*
- *When N is a body node, let the guard nodes which are brothers of N be $N_1,...,N_m$. Also let $l(parent(N))$ be $p(v_1,...,v_n)$ and $t_1,...,t_{m'}$ be all the arguments of $l(N_1)$, ..., $l(N_m)$. Assume that $[t'']^*_{e|V}$ is the greatest element of $[\langle v_1,\cdots,v_n,t_1,\cdots,t_{m'}\rangle]^*_e$. If $[t'']^*_{e'|V}$ is not the greatest element of $[\langle v_1,\cdots,v_n,t_1,\cdots,t_{m'}\rangle]^*_{e'}$, the goal is suspended. Otherwise, the goal is not suspended.* □

Definition 3.6 *A leaf node N is called ready under the environment e if one of the following conditions is satisfied.*

1. *$l(N)$ is $t = t'$ and is not suspended under the environment e.*
2. *$l(N)$ is $p(\cdots)$ where p is a user defined predicate symbol.*

A node to which a head is attached is called ready if the node has no child which is a guard node. □

Definition 3.7 *The domain S is defined to be $\{(T,e)|$ T is an execution tree, $e \in E\}$.* □

An element of S represents the execution status of a GHC program.

The next definition illustrates the goal reduction strategy of GHC. In this paper, for convenience, we define the semantics of GHC so that each unification process that would fail will be suspended. With this definition, there is no change in the semantics of success programs. Of course, by modifying *1* of Definition 3.8 slightly, we can obtain the semantics in which each failure unification actually fails within finite time.

Definition 3.8 *Let T and T' be execution trees and e be a successful environment. The relation $\stackrel{reduce}{\Longrightarrow}$ ($\in S \times S$) is defined as follows:*

1. *When T has a leaf N to which a ready goal $t = t'$ is attached:*

$$(T,e) \stackrel{reduce}{\Longrightarrow} (remove(T,N), e \sqcup \{(t,t')\}^c)$$

is satisfied if $e \sqcup \{(t,t')\}^c$ is a successful environment.

2. *Suppose that T has a leaf N to which a goal $p(t_1,...,t_m)$, where p is a user defined predicate symbol, is attached and there are n clauses:*

$$p(v_{1,1},...,v_{1,m}) :- G_{1,1},...,G_{1,i_1}|B_{1,1},...,B_{1,j_1}.$$
$$\vdots$$
$$p(v_{n,1},...,v_{n,m}) :- G_{n,1},...,G_{n,i_n}|B_{n,1},...,B_{n,j_n}.$$

whose heads contain p [3].

$$(T, e) \stackrel{reduce}{\Longrightarrow} (T_n, e \sqcup \{(\langle t_1, ..., t_m \rangle, \langle v_{k,1}, ..., v_{k,m} \rangle)\}_{1 \le k \le n}^{c})$$

is satisfied if T_n *is defined by the following equations:*

$$T_0 = add(T, N, p(v_{1,1}, ..., v_{1,m}), ..., p(v_{n,1}, ..., v_{n,m}))$$
$$T_k = add(T_{k-1}, child(N, k), G_{k,1}, ..., G_{k,i_k}, B_{k,1}, ..., B_{k,j_k}) \quad (1 \le k \le n)$$

Also extra information must be stored on the guard nodes of T_n.

3. *When* T *has a node* N *to which a head* H *is attached and which has no guard child and* n *body children* $(0 \le n)$*:*

$$(T, e) \stackrel{reduce}{\Longrightarrow} (replace(T, N, child(N, 1), ..., child(N, n)), e)$$

is satisfied. In this case, if N *is a guard node in* T*, the nodes* $child(N, 1)$*, ..., and* $child(N, n)$ *must become guard nodes.* □

Each element of the semantic domain introduced in the next definition represents an execution history of a GHC program.

Definition 3.9 *The domain* $S^* \subseteq \mathbf{N} \rightarrow S$ *is the set of the elements, say* s^**, satisfying the following conditions:*

1. *If* $s^*(n) = \perp_S$ *for some* n*,* $s^*(m) = \perp_S$ $(n \le m)$.
2. *If* $s^*(0) \neq \perp_S$*,* $s^*(0) = (T, \perp_E)$ *such that* T *consists of the root and its children.*
3. *If neither* $s^*(n)$ *nor* $s^*(n+1)$ *is equal to* $\perp_S$ $(0 \le n)$*,* $s^*(n) \stackrel{reduce}{\Longrightarrow} s^*(n+1)$
4. *If* $s^*(n) = (T, e)$ $(0 \le n)$ *and* T *contains a ready node* N *under* e*, there exists* m $(> n)$ *such that* $s^*(m)$ *does not contain* N *or* N *in* $s^*(m)$ *is not ready.* □

By the last condition in the above definition, S cannot be a complete domain.

Definition 3.10 *Suppose that* s_1^* *and* s_2^* *are elements of* S^**. We define a partial order relation* $\sqsubseteq$ *on* S^* *such that* $s_1^* \sqsubseteq s_2^*$ *is satisfied if and only if:*

1. $s_1^* = s_2^*$*, or*
2. $\exists n$ *s.t.* $s_1^*(m) = s_2^*(m)$ *if* $m < n$ *and* $s_1^*(m) = \perp_S$ *otherwise.* □

Intuitively speaking, $s_1^* \sqsubseteq s_2^*$ is satisfied if and only if the execution history represented by s_1^* is an initial segment of the one represented by s_2^*.

Definition 3.11 $D \subseteq 2^{S^*}$ *is the set of the elements, say* d*, which satisfies the following conditions:*

[3] Some of the clauses may not have guards and/or bodies

1. *If s_1^* is an element of d and $s_2^* \sqsubseteq s_1^*$, s_2^* is also an element of d.*
2. *Suppose that $\{s_i^*\}_{i\in\mathbf{N}} \subset d$ and $s_1^* \sqsubseteq s_2^* \sqsubseteq ... \sqsubseteq s_i^* \sqsubseteq ....$ $\sqcup_{i\in\mathbf{N}} s_i^*$ is an element of d if it exists.* □

Notice that D can be regarded as a power domain. Each element of D represents a set of possible execution histories.

Definition 3.12 *Suppose that $X \subseteq S^*$.*

$$X^c \stackrel{def}{\equiv} \sqcap\{d \in D | X \subseteq d\} \quad \square$$

Theorem 3.13 *D is a complete lattice under the set inclusion order.*

Proof: *Let I be an index set of arbitrary cardinality. Suppose $d_i \in D$ $(i \in I)$. We can easily prove that $\sqcap_{i\in I} d_i = \cap_{i\in I} d_i$ and $\sqcup_{i\in I} d_i = (\cup_{i\in I} d_i)^c$* □

Next, we define transformation functions. The first one transforms an execution history to the set of possible execution histories in the next stage.

Definition 3.14 *$Trans : S^* \to D$ is defined as follows:*

1. *If $s^*(n) \neq \perp_S$ for all n, $Trans(s^*) = \{s^*\}^c$.*
2. *Otherwise, let n be the natural number such that $s^*(n) = (T, e) \neq \perp_S$ and $s^*(n+1) = \perp_S$.*
   (a) If T has a ready node under e,

$$Trans(s^*) = \left\{ s_1^* \;\middle|\; \begin{array}{l} s^*(m) = s_1^*(m) \text{ if } m \leq n, \\ s^*(n) \stackrel{reduce}{\Longrightarrow} s_1^*(n+1), \text{ and} \\ s_1^*(m) = \perp_S \text{ if } m > n+1 \end{array} \right\}^c$$

   (b) If T does not have any ready node under e, $Trans(s^*) = \{s^*\}^c$.

Definition 3.15 *Goal is a set of the goal clauses for P.* □

For simplicity, we assume that each functor/predicate symbol occurring in a goal clause in *Goal* also appears in P.

Definition 3.16 *$Trans^* : Goal \to D \to D$ is defined as follows: For all $g \in Goal, d \in D$,*

$$Trans^*(g)(d) = (\bigsqcup_{s^*\in d} Trans(s^*)) \sqcup d \sqcup \{s_g^*\}^c$$

where s_g^* satisfies that:

1. $s_g^*(0)$ *is the pair of the initial execution tree for the goals* g *and the initial environment* $\perp_E = \{(t,t)|t \in Term_{F,V}\}$.
2. $s_g^*(n) = \perp_S (n \geq 1)$. □

Theorem 3.17 $Trans^*(g)$ *is a continuous function.*

Proof: *Let* X *be a subset of* D. *We prove that* $\sqcup Trans^*(g)(X) = Trans^*(g)(\sqcup X)$. *Obviously,* $Trans^*(g)$ *is a monotonic and so* $\sqcup Trans^*(g)(X) \sqsubseteq Trans^*(g)(\sqcup X)$. *On the other hand,*

$$\begin{aligned} Trans^*(g)(\sqcup X) &= (\bigsqcup_{s^* \in \sqcup X} Trans(s^*)) \sqcup (\sqcup X) \sqcup \{s_g^*\}^c \\ &= (\bigsqcup_{s^* \in \sqcup X - \cup X} Trans(s^*)) \sqcup (\bigsqcup_{s^* \in \cup X} Trans(s^*)) \sqcup (\sqcup X) \sqcup \{s_g^*\}^c \end{aligned}$$

Obviously, $Trans^*(g)(\sqcup X) \sqsupseteq \sqcup X$, $\{s_g^*\}^c$, $\bigsqcup_{s^* \in \cup X} Trans(s^*)$. *Therefore, it is sufficient for us to prove* $Trans^*(g)(\sqcup X) \sqsupseteq \bigsqcup_{s^* \in \sqcup X - \cup X} Trans(s^*)$.

We can prove that each element s^* *of* $\sqcup X - \cup X$ *is a limit element in the sense that* $s^*(n) \neq \perp_S (0 \leq n)$ [4] *and thus* $Trans(s^*) = \{s^*\}^c$. *This implies* $\bigsqcup_{s^* \in \sqcup X - \cup X} Trans(s^*) = \sqcup X - \cup X$, *which is obviously less than or equal to* $Trans^*(g)(\sqcup X)$. □

Since D is a complete lattice and $Trans^*(g)$ is a continuous function, $Trans^*(g)$ has the least fixed point $fix(Trans^*(g))$ in D satisfying:

$$fix(Trans^*(g)) = \bigsqcup_{n=1}^{\infty} (Trans^*(g))^n(\perp_D)$$

Of course, $fix(Trans^*(g))$ is extravagant as the semantics of the initial goal g. We do not need the possible execution trees but just the possible variable binding environments concerning the arguments of g.

4 A Fixed Point Semantics of CP

In this section, we develop a fixed point semantics of CP according to the description of [Shapiro83a]. This semantics is just a modification of the one in Section 3.

The most significant differences of GHC and CP are

1. CP is a multiple environment language, and
2. CP employs the read only annotation ?.

In order to cope with multiple environments, we modify the definition of an execution tree (i.e. Definition 3.4) so that the pair of a literal and an environment is attached to each node of an even depth. These literal and environment represent the head of some program clause and the variable binding environment for reducing the clause, respectively.

[4] Such an element represents an infinite computation.

For simplicity, we assume that the pair of a dummy literal and the outer most environment is attached to the root node.

In order to deal with the read only annotation, we modify the set V such that $V = V' \cup \{v? | v \in V'\}$ where V' is a set of countably many variables. That is, we consider v and $v?$ are distinct variables. We call each element in V' an *ordinary variable* and each element in $V - V'$ an *annotated variable*. In this case, the suspension rule defined in Definition 3.5 must be modified as the next one.

Definition 4.1 *Let N be a leaf of some execution tree to which a goal $t = t'$ is attached and which is a guard node or a child of the root. Assume that $l(parent(N))$ is $\langle H, e\rangle$ where H is a literal and e is a successful environment. The goal $t = t'$ is suspended if and only if, for some variable $v?$ and non-variable term t'', $[v?]^*_{e|V}$ and $[t'']^*_{e|V}$ are the greatest elements of $[t]^*_e$ and $[t']^*_e$.* □

According to the above definition, each goal in a body part is not executed until the clause which contains the goal is committed.

Definition 3.6 need just a slight modification. In Definition 3.6, whether or not a node is ready depends on an environment. However, currently, an execution tree contains the environment of each process. Therefore, this concept does not depend on an environment. Also Definition 3.7 should be modified. In this section, the domain S is the set of the execution trees.

Lastly, we have to modify the reducibility relation (i.e. Definition 3.8) as the next one.

Definition 4.2 *Let T be an execution tree. The relation $\stackrel{reduce}{\Longrightarrow}$ is defined as follows:*

1. *Suppose that T has leaf N to which a ready goal $t = t'$ is attached. There are three cases:*

   - *When, for some (ordinary and/or annotated) variables v and v', $[v]^*_{e|V}$ and $[v']^*_{e|V}$ are the greatest elements of $[t]^*_e$ and $[t']^*_e$:*

     $$T \stackrel{reduce}{\Longrightarrow} remove(T, N)[N_1/\langle H_1, e_1 \sqcup \{(t, t')\}^c\rangle, ..., N_n/\langle H_n, e_n \sqcup \{(t, t')\}^c\rangle]$$

     is satisfied, where $N_1, ..., N_n$ are the nodes $parent(N)$ and its offsprings of even depths, and $l(N_k)$ is $\langle H_k, e_k\rangle$ $(1 \leq k \leq n)$.

   - *When, for some (ordinary or annotated) variable v and some term $f(t_1, ..., t_n)$, $[v]^*_{e|V}$ and $[f(t_1, ..., t_n)]^*_{e|V}$ are the greatest elements of $[t]^*_e$ and $[t']^*_e$:*

     $$T \stackrel{reduce}{\Longrightarrow} remove(T, N)[N_1/\langle H_1, e_1 \sqcup \{(t, t')\}^c \sqcup e'\rangle, ..., N_n/\langle H_n, e_n \sqcup \{(t, t')\}^c \sqcup e'\rangle]$$

     is satisfied, where $N_1, ..., N_n$ are the nodes $parent(N)$ and its offsprings of even depths, $l(N_k)$ is $\langle H_k, e_k\rangle$ $(1 \leq k \leq n)$, and e' is:

     $$\{(v', v'?) | [v']^*_{e|V} \text{ is the greatest element of } [v]^*_e\}^c$$

- *When, for some terms* $f(t_1,...,t_n)$ *and* $f(t'_1,...,t'_n)$*, the greatest elements of* $[t]^*_e$ *and* $[t']^*_e$ *are* $[f(t_1,...,t_n)]^*_{e|V}$ *and* $[f(t'_1,...,t'_n)]^*_{e|V}$*:*

$$T \stackrel{reduce}{\Longrightarrow} replace(T, N, N_1, ..., N_n)$$

is satisfied, where $N_1,...,N_n$ *are the newly created nodes to which* $t_1 = t'_1$*, ..., and* $t_n = t'_n$*, respectively, are attached.*

2. *Suppose that* T *has a leaf* N *to which a goal* $p(t_1,...,t_m)$*, where* p *is a user defined predicate symbol, is attached and there are* n *clauses:*

$$\begin{aligned} p(v_{1,1},...,v_{1,m}) &:- G_{1,1},...,G_{1,i_1} | B_{1,1},...,B_{1,j_1}. \\ &\vdots \\ p(v_{n,1},...,v_{n,m}) &:- G_{n,1},...,G_{n,i_n} | B_{n,1},...,B_{n,j_n}. \end{aligned}$$

whose heads contain p*. Let* $\langle H, e\rangle$ *be* $l(parent(N))$*.* $T \stackrel{reduce}{\Longrightarrow} T_n$ *is satisfied if* T_n *is defined by the following equations:*

$$\begin{aligned} T_0 &= add(T, N, \langle p(v_{1,1},...,v_{1,m}), e \sqcup e_1\rangle, ..., \langle p(v_{n,1},...,v_{n,m}), e \sqcup e_n\rangle) \\ &\quad where\ e_k = \{(t_{k'}, v_{k,k'})\}^c_{1\le k'\le m} \quad (1 \le k \le n) \\ T_k &= add(T_{k-1}, child(N,k), G_{k,1},...,G_{k,i_k}, B_{1,k},...,B_{k,j_k}) \quad (1 \le k \le n) \end{aligned}$$

Also extra information must be stored on the guard nodes of T_n*.*

3. *When* T *has a node* N *to which a head-environment pair* $\langle H, e\rangle$ *is attached and which has no guard child and* n *body children* $(0 \le n)$*:*

$$T \stackrel{reduce}{\Longrightarrow} replace(T, N, child(N,1), ..., child(N,n))[N_1/\langle H_1, e \sqcup e_1\rangle, ..., N_n/\langle H_n, e \sqcup e_n\rangle]$$

is satisfied, where $N_1,...,N_n$ *are the nodes which are* $parent(parent(N))$ *and its offsprings of even depths and* $l(N_k)$ *is* $\langle H_k, e_k\rangle$*. Also, if* N *is a guard node in* T*, the nodes* $child(N,1)$*, ..., and* $child(N,n)$ *must become guard nodes.* □

The above definition is almost same as Definition 3.8. However, *1* is more complex and operational than that in Definition 3.8. The reason is that CP employs the read only annotation which has no *logical foundation*.

After that, with the power domain technique, we can develop the fixed point semantics of CP in the same manner in Section 3.

5 Conclusion

In this paper, we introduce two kinds of semantic domains, one for variable binding environments and the other for goal reduction strategies. The former domain (i.e. E) is general enough to be applicable to the semantics description of almost all logic (or unification based)

programming language. In contrast, the latter domains (i.e. S and D in Section 3 and 4) may be modified when applying to other parallel logic programming languages such as Parlog. However, even in such cases, the power domain technique employed in this paper can be effectively used.

Based on the former domain, we have developed the semantics of unification. This semantics allows us to hide an actual implementation of unification processes behind mathematical devices. In this respect, our approach is not operational. However, we describe goal reduction strategies in an operational manner. Consequently, our approach is a mixture of denotational and operation descriptions. The language features which are based on logic (or mathematics) are described in a denotational way and the other ones are described in an operational way.

Comparing our semantics descriptions of GHC and CP, we feel that the introduction of the read only annotation makes the semantics of CP complicated. On the other hand, the non-logical features of GHC (i.e. suspension and guard mechanisms) seems simpler and more logical than the read only annotation.

Acknowledgements

The author would like to thank Prof. Yonezawa for his kind comments and suggestions.

References

[Apt82] K. R. Apt and M. H. Emden: Contributions to Theory of Logic Programming, *Journal of the ACM*, Vol. 29, No. 3, pp. 841–862, 1982.

[Beckman86] L. Beckman: Towards a Formal Semantics for Concurrent Logic Programming Languages, *Third International Conference on Logic Programming*, Lecture Notes in Computer Science, Vol. 225, pp. 335–349, Springer-Verlag, 1986.

[Clark86] K. Clark and S. Gregory: PARLOG: Parallel Programming in Logic, *ACM Trans. on Programming Languages and Systems*, Vol. 8, No. 1, pp. 1–49, 1986.

[Clinger81] W. D. Clinger: Foundations of Actor Semantics, (Ph.D. thesis), Dept. of Math., M.I.T, 1981.

[Colmerauer82] A. Colmerauer: Prolog and Infinite Trees, *Logic Programming* edited by K. L. Clark and S.-Å. Tärnlund, Academic Press, 1982.

[Emden76] M. H. Emden and R. A. Kowalski: The Semantics of Predicate Logic as a Programming Language, *Journal of the ACM*, Vol. 23, No. 4, pp. 733–742, 1976.

[Lassez84] J.-L. Lassez and M. J. Maher: Closures and Fairness in the Semantics of Programming Logic, *Theoretical Computer Science*, Vol. 29, pp. 167–184, 1984.

[Lloyd84] J. W. Lloyd: *Foundations of Logic Programming*, Springer-Verlag, 1984.

[Robinson65] J. A. Robinson: A Machine-Oriented Logic Based on the Resolution Principle, *Journal of the ACM*, Vol. 12, No. 1, pp. 23–41, 1965.

[Saraswat85] V. A. Saraswat: Partial Correctness Semantics for CP[↓,—,&], *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, Vol. 206, pp. 347–368, Springer-Verlag, 1985.

[Shapiro83a] E. Shapiro: A Subset of Concurrent Prolog and Its Interpreter, *Technical Report TR-003*, Institute for New Generation Computer Technology, 1983.

[Shapiro83b] E. Shapiro and A. Takeuchi: Object Oriented Programming in Concurrent Prolog, *New Generation Computing*, Vol. 1, No. 1 (1983), pp. 25–48

[Ueda85] K. Ueda: Guarded Horn Clauses, *Proceedings of the Logic Programming Conference'85*, Tokyo, pp. 225–236, 1985.

[Ueda86] K. Ueda: On the Operational Semantics of Guarded Horn Clauses, *RIMS Kou-kyu-Roku*, Research Institute for Mathematical Sciences, Kyoto Univ., pp. 263–283, 1986.