TR-406

A. Declarative Semantics of Parallel Logic Programs with Perpetual Processes

by
M. Murakami

June, 1988

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Declarative Semantics of Parallel Logic Programs with Perpetual Processes

Masaki Murakami

Institute for New Generation Computer Technology
21F, Mita Kokusai Building
1-4-28 Mita, Minato-ku, Tokyo 108 Japan

June 10, 1988

Abstract

A declarative semantics of a parallel programming language based on Horn logic such as Flat GHC is presented. The domain of input/output (I/O) histories is introduced. The denotation of a program is defined as a set of I/O histories. The notion of truth is redefined for goal clauses and sets of guarded clauses. The semantics of a program is defined as the maximum model of the program. We also show that the semantics is characterized as the greatest fixpoint of the function obtained from the program. The properties of programs which contain perpetual computation controlled by guard-commit mechanisms can be discussed using the semantics.

1 Introduction

In recent years, several parallel programming languages based on Horn logic have been investigated. Examples are PARLOG [Clark 86], Concurrent Prolog [Shapiro 86] and GHC [Ueda 88]. In such languages, the notion of pro-

cesses which execute infinite computations controlled by guard-commit mechanisms communicating with other processes using input/output streams can be represented naturally. Several results on the formal semantics of these languages are reported [Saraswat 85, 87, Ueda 86, Shibayama 87, Takeuchi 86]. However, these results are based on the operational approach. Thus, they should be considered as a formal specification of the language processing system. In order to give a logical base for program verification methods or transformation methods, a kind of declarative semantics is expected.

In pure Horn logic programming languages, the result for declarative semantics based on the least fixpoint is reported in [Apt 82, Lloyd 84]. In this approach, the denotation of a program is given as the minimum model of the set of Horn clauses, in other words, the set of unit clauses which is equivalent to the program. The set of unit clauses is characterized as the least fixpoint of the function obtained from the set of definite clauses. In this approach, we can characterize the set of solutions as the logical consequences of the program independently from the execution mechanisms. This approach is one of the best ways of appreciating the clarity of logic programs. Extensions of this approach for programs which contain infinite computations are also reported in [Lloyd 84, Sakakibara 85].

However, these results are reported for pure Horn logic languages. They cannot be applied to parallel languages which contain the notion of a guard-commit mechanism directory. A model theory must be reconstructed for Horn logic with the commit operator. Thus several extensions are reported for such languages [Levi 87, 88, Falaschi 88a]. [Levi 88] discusses the semantics of Flat GHC programs as the sets of guarded atoms. A guarded atom is a guarded clause such that all atoms in the guard part and the body part are unifications. For example,

$$p(X_1, \ldots, X_n, Y_1, \ldots, Y_m) :- \\ X_1 = \tau_1, \ldots, X_n = \tau_n | Y_1 = \theta_1, \ldots, Y_m = \theta_m.$$

They can be considered as unit clauses of Flat GHC.

However, in this approach, the guarded atom describes only the relation between the input substitutions and the compute substitutions which are obtained when the goal succeeds. It is difficult to discuss the infinite computation of the program only with such relation. As [Takeuchi 86] reported, there are two programs which cannot be distinguished only by relations of

input and output substitutions, and output different results when they are executed in parallel with other processes. Thus, the information for the intermediate result of the computation is necessary to discuss the semantics of such programs.

This paper introduces a new declarative semantics for Flat GHC programs which contain perpetual processes. The notion of *input/output (I/O) history* is introduced instead of the notion of the guarded atom. Intuitively an I/O history denotes an example of a computation path of a program which is generated when the program is executed without any failure or deadlock. We define the notion that a goal clause or a set of guarded clauses is true wrt a set of I/O histories. The semantics of a program is defined as the maximum set of I/O histories which makes the program true, in other words, the maximum model of the program.

The notion of a true goal clause wrt the model of a program does not necessarily mean that the goal clause succeeds on the program. That is, not only all successful goal clauses are true but also goals which do not succeed finitely but can be executed infinitely without failure or deadlock are true. A goal clause with a goal which suspends can also be true if the goal can be activated with some input from other processes.

This paper also shows that the semantics of a program can be characterized as the greatest fixpoint of the function obtained from the program.

2 Guarded Stream

This section introduces the notion of the the *guarded stream*. For simplicity we only consider programs on the domain of lists of {`a, b`} .

Def. 1

Let *Var* be a set of variables, *Fun*= {`a, b, nil, cons`} be a set of function symbols. Each element of *Fun* has its arity. The arity of `a, b` and `nil` is 0, and the arity of `cons` is 2.

Def. 2

Let *Terms* be the set of terms defined as follows.

(i) if $\tau \in Var$ or $\tau \in \{\mathtt{a}, \mathtt{b}, \mathtt{nil}\}$, then $\tau \in Terms$.

(ii) if $\tau_1, \tau_2 \in Terms$, then $\mathtt{cons}(\tau_1, \tau_2) \in Terms$.

Def. 3

A term τ is said to be *simple*, when $\tau \in Var$, $\tau \in \{\mathtt{a},\ \mathtt{b},\ \mathtt{nil}\}$ or τ has the form of $\mathtt{cons}(X, Y)$, and X and Y are different variables.

Def. 4

A mapping, σ: $Var \rightarrow Terms$ is called a *substitution* if it satisfies the following condition:

$$\text{If } \Sigma = \{X | \sigma X \neq X, X \in Var\}, \text{ then } \Sigma \text{ is a finite set.}$$

We expand the domain of substitutions from *Var* to *Terms*.

Def. 5

For each $\tau \in Terms$, $\sigma\tau$ is recursively defined as follows.

$$\sigma\tau = \begin{cases} \sigma X & \text{if } \tau \text{ is } X \in Var \\ \tau & \text{if } \tau \in \{\mathtt{a},\ \mathtt{b},\ \mathtt{nil}\} \\ \mathtt{cons}(\sigma\tau_1, \sigma\tau_2) & \text{if } \tau = \mathtt{cons}(\tau_1, \tau_2), \tau_1, \tau_2 \in Terms \end{cases}$$

Def. 6

Let τ be a simple term and $X \in Var$.

$$X = \tau$$

is a *simple substitution form* or a *substitution form* simply. $X = X$ is denoted *true*.

A substitution σ is denoted using a finite set of simple substitution forms, for example,

$$\sigma = \{\mathtt{X = cons(Y,\ Z),\ Y = a}\}.$$

$\mathtt{cons}(X, Y)$ is denoted $[X|Y]$ and $\mathtt{nil}$ is denoted [ ].

Def. 7

Let σ be a set of simple substitution forms. If σ is a substitution or equal to $\bigcup_{k-\infty} \theta_k$ defined below for some substitution θ, then σ is be an *ω−substitution*.

$$\begin{aligned} &\theta_0 = \theta \\ &\theta_{k+1} = \theta_k \cup \\ &\qquad \{X = \tau | X \text{ occurs in } \tau' \text{ for some } (Y = \tau') \in \theta_k, \\ &\qquad\ (X = \tau'') \notin \theta_k \text{ and no variables occurring in } \tau \\ &\qquad\ \text{occur in any element of } \theta_k\} \end{aligned}$$

A $\omega-$substitution defines a mapping from a term to an infinite term.

Def. 8

Let V be a set of variables ($V \subset Var$), and Σ be the set defined from a mapping σ as **Def. 4**. If $\Sigma \subset V$, then σ is *restricted to* V. If $\Sigma \cap V = \phi$, then σ is *invariant on* V.

The notion of I/O history introduced in this paper corresponds to the notion of element of the Herbrand bases for pure Horn logic programs. I/O history is an extension or modification of a guarded atom of [Levi 88]. An I/O history is denoted as follows with head part H, which denotes a form of a process, and the body part GU, which denotes a trace of inputs and outputs of the process:

$$H : -GU.$$

H is defined in the next section. GU is a set of tuples $< \sigma | U_b >$ where σ is a substitution which is expressed in the form of a set of simple substitution forms, and U_b is a formula which represents an execution of a unification in the body part of some clause. Intuitively, $< \sigma | U_b >$ means that the arguments of the process are instantiated with σ, then unification U_b is executed. For instance, in the following program:

```
p1(X,Y) :- X = [A|X1], A = a | Y = [B|Y1], B = b, p1(X1,Y1).
p1(X,Y) :- X = [B|X1], B = b | Y = [A|Y1], A = a, p1(X1,Y1).
```

The following is an example of I/O history which denotes the computation such that `p1` reads `a` in input stream `X` first, writes `b` in output stream `Y`, then reads `b` and writes `a`.

$$\begin{aligned} \texttt{p1(X,Y)} : -\{ & < \{\texttt{X} = [\texttt{A}|\texttt{X1}], \texttt{A} = \texttt{a}\} | \texttt{Y} = [\texttt{B}|\texttt{Y1}] >, \\ & < \{\texttt{X} = [\texttt{A}|\texttt{X1}], \texttt{A} = \texttt{a}\} | \texttt{B} = \texttt{b} >, \\ & < \{\texttt{X} = [\texttt{A}|\texttt{X1}], \texttt{A} = \texttt{a}, \texttt{X1} = [\texttt{B1}|\texttt{X2}], \texttt{B1} = \texttt{b}\} | \\ & \quad \texttt{Y1} = [\texttt{A1}|\texttt{Y2}] >, \\ & < \{\texttt{X} = [\texttt{A}|\texttt{X1}], \texttt{A} = \texttt{a}, \texttt{X1} = [\texttt{B1}|\texttt{X2}], \texttt{B1} = \texttt{b}\} | \\ & \quad \texttt{A1} = \texttt{a} >, \ldots\} \end{aligned}$$

An I/O history of a process H represents a possible execution of the

process. Thus, there exist different I/O histories for different executions which commit to different clauses. There may be different I/O histories for different schedulings.

The body part of an I/O history represents a normal execution of Flat GHC programs, thus GU is well founded with the partial order of execution, namely, for any $< \sigma_1|U_{b1} >, < \sigma_2|U_{b2} > \in GU$, if $\sigma_1 \subset \sigma_2$, then U_{b1} is executable before U_{b2}.

GU has several characteristic which correspond to the properties of normal executions of GHC programs. In the rest of this section, the notion of guarded stream is introduced which characterizes the normal executions of GHC programs.

Def. 9

Let τ be a simple term and $X \in Var$.

$$X? = \tau$$

is a *simple test form* or a *test form* simply.

Def. 10

For a substitution σ, $X \in Var$, and a simple term τ, $< \sigma|uni(X, \tau) >$ is *a guarded unification*, where $uni(X, \tau)$ denotes $X = \tau$ or $X? = \tau$. σ is the *guard part* of $< \sigma|uni(X, \tau) >$ and $uni(X, \tau)$ is the *active part*.

Intuitively, if $uni(X, \tau)$ is a substitution form, it denotes a unification which actually instantiates X, and if it is a test form, it denotes a test unification.

Def. 11

Let $< \sigma|U >$ be a guarded unification. $| < \sigma|U > |$ is the set of substitution forms or test from defined as following.

$$| < \sigma|U > | = \{U\} \cup \sigma$$

Def. 12

Let GU be a set of guarded unifications. For $< \sigma_1|u_1 >, < \sigma_2|u_2 > \in GU$,

$$< \sigma_1|u_1 > \prec < \sigma_2|u_2 >$$

holds if and only if $\sigma_1 \subset \sigma_2$ and $\sigma_1 \neq \sigma_2$.

It is easy to show that $\prec$ is a well founded ordering.

Def. 13

A set of guarded unifications GU is said to be a *guarded stream* if the following are true.

(i) For any $< \sigma_1|U_1 >, < \sigma_2|U_2 > \in GU$, if $< \sigma_1|U_1 > \neq < \sigma_2|U_2 >$ and U_1 and U_2 have same variable on their left hand side then, either U_1 or U_2 is a test form and their right hand sides are identical. Furthermore, if U_1 is a substitution form and U_2 is a test form then

$$< \sigma_2|U_2 > \prec < \sigma_1|U_1 >$$

does not hold.
(ii) If $< \sigma|U > \in GU$, then $(X = \tau) \notin \sigma$ for any $< \theta|X = \tau > \in GU$.
(iii) For any $< \theta|X? = \tau > \in GU$, if $\tau \neq \tau'$, then $(X = \tau') \notin \sigma$ for $< \sigma|U > \in GU$.
(iv) For any $< \sigma_1|U_1 >, < \sigma_2|U_2 > \in GU$, if $(X = \tau) \in \sigma_1$ and $(X = \tau') \in \sigma_2$, then $\tau = \tau'$.

Conditions (i) to (iv) mean that all variable in GHC programs are logical variables and if they are instantiated, the values are never changed.

Example 1

The following are examples of guarded streams. GU_1 represents an execution of the process which takes a sequence of "a"s in Z and outputs a sequence of "b"s in Y. GU_2 represents an execution of the process which takes a sequence of "a"s in X and a sequence of "b"s in Y respectively, and merges them and outputs the result to Z.

$$GU_1 = \{gu_{1i}(1 \leq i)\},$$

$gu_{11} =< \{\mathtt{Z} = [\mathtt{A}|\mathtt{Z1}], \mathtt{A} = \mathtt{a}\}|\mathtt{Y} = [\mathtt{B}|\mathtt{Y1}] >$
$gu_{12} =< \{\mathtt{Z} = [\mathtt{A}|\mathtt{Z1}], \mathtt{A} = \mathtt{a}\}|\mathtt{B} = \mathtt{b} >$
$gu_{13} =< \{\mathtt{Z} = [\mathtt{A}|\mathtt{Z1}], \mathtt{A} = \mathtt{a}, \mathtt{Z1} = [\mathtt{A1}|\mathtt{Z2}], \mathtt{A1} = \mathtt{a}\}|\mathtt{Y1} = [\mathtt{B1}|\mathtt{Y2}] >$
$gu_{14} =< \{\mathtt{Z} = [\mathtt{A}|\mathtt{Z1}], \mathtt{A} = \mathtt{a}, \mathtt{Z1} = [\mathtt{A1}|\mathtt{Z2}], \mathtt{A1} = \mathtt{a}\}|\mathtt{B1} = \mathtt{b} >$
.....

$$GU_2 = \{gu_{2j}(1 \leq j)\}$$

$gu_{21} =< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}\}|\mathtt{Z} = [\mathtt{A}|\mathtt{Z1}] >$
$gu_{22} =< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}\}|\mathtt{A} = \mathtt{a} >$
$gu_{23} =< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}, \mathtt{Y} = [\mathtt{B}|\mathtt{Y1}], \mathtt{B} = \mathtt{b}\}|\mathtt{Z1} = [\mathtt{A1}|\mathtt{Z2}] >$
$gu_{24} =< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}, \mathtt{Y} = [\mathtt{B}|\mathtt{Y1}], \mathtt{B} = \mathtt{b}\}|\mathtt{A1} = \mathtt{b} >$
$gu_{25} =< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}, \mathtt{Y} = [\mathtt{B}|\mathtt{Y1}], \mathtt{B} = \mathtt{b}, \mathtt{Y1} = [\mathtt{B1}|\mathtt{Y2}], \mathtt{B1} = \mathtt{b}\}|$
$\mathtt{Z2} = [\mathtt{B2}|\mathtt{Z3}] >$
$gu_{26} =< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}, \mathtt{Y} = [\mathtt{B}|\mathtt{Y1}], \mathtt{B} = \mathtt{b}, \mathtt{Y1} = [\mathtt{B1}|\mathtt{Y2}], \mathtt{B1} = \mathtt{b}\}|$
$\mathtt{B2} = \mathtt{b} >$
.....

□

The following notion is defined to obtain the guarded stream representing the computation of a goal clause from the guarded streams which represent the computation of each goal in the goal clause.

Def. 14

Let $GU_1, \ldots, GU_n$ be guarded streams, and $Gu_k (1 \leq k)$ be as follows:

$$Gu_1 = \{< \sigma | U > | \exists i, \exists < \sigma | U > \in GU_i, \\ \forall U' \in \sigma, \forall j, < \sigma' | U' > \notin GU_j\}$$

$$Gu_{k+1} = Gu_k \cup \\ \{< \sigma | U > | \exists i, \exists < \sigma' | U > \in GU_i, \forall U' \in \sigma', \\ ((\forall j, < \sigma'' | U' > \notin GU_j) \vee < \sigma'' | U' > \in Gu_k) \wedge \\ \sigma = (\sigma' - \{U' | < \sigma'' | U' > \in Gu_k\}) \cup \\ \{U'' | U'' \in \sigma'', < \sigma'' | U' > \in Gu_k\}\}$$

and let GU be as follows.

$$GU = \bigcup_{k \to \infty} Gu_k$$

If GU is a guarded stream and if

$$\{U | < \sigma | U > \in GU\} = \{U | \exists i < \sigma | U > \in GU_i\}$$

then GU is a *synchronized merge of* $GU_1, \ldots, GU_n$, and is denoted:

$$GU_1 \| \ldots \| GU_n.$$

If $n = 1$, then the synchronized merge can always be defined and it is equal to GU_1 itself.

In **Def. 14**, for $< \sigma | U > \in GU_i$, if $< \sigma | U > \in Gu_1$ then that means U waits σ from outside of $GU_1 \| \ldots \| GU_n$ and waits nothing from $GU_j (j \neq i)$. If $< \sigma | U > \in Gu_{k+1}$ then that means U waits some inputs from outside of $GU_1 \| \ldots \| GU_n$ and waits that U' is executed in some $GU_j (j \neq i)$ such that it is already found that it waits σ'' from outside. If U' waits $U'' \in \sigma''$ from outside of $GU_1 \| \ldots \| GU_n$, then U also waits U''.

Consider following guarded streams.

$$GU_1 = \{< \{\mathtt{X} = \mathtt{a}\} | \mathtt{Y} = \mathtt{b} >\}$$
$$GU_2 = \{< \{\mathtt{Y} = \mathtt{b}\} | \mathtt{X} = \mathtt{a} >\}$$

They represent computations of the `g1(X, Y)` and `g2(Y, X)` where,

```
g1(X, Y) :- X = a | Y = b.
g2(Y, X) :- Y = b | X = a.
```

In this case,

$$\{U | < \sigma | U > \in GU\} = \phi.$$

On the other hand,

$$\{U | \exists i < \sigma | U > \in GU_i\} = \{\mathtt{X} = \mathtt{a}, \mathtt{Y} = \mathtt{b}\}.$$

Thus $GU_1 \| GU_2$ cannot be defined. It is impossible to obtain the guarded stream which represents the computation of goal clause `g1(X, Y), g2(Y, X)` from GU_1 and GU_2. In fact, neither `X` nor `Y` is instantiated by execution of `g1(X, Y), g2(Y, X)`.

Example 2

Let GU_1 and GU_2 be the same as **Example 1**.

$Gu_1 = \{gu_{21}, gu_{22}\}$

$$Gu_2 = Gu_1 \cup \{< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}\}|\mathtt{Y} = [\mathtt{B}|\mathtt{Y1}] >,$$
$$< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}\}|\mathtt{B} = \mathtt{b} >\}$$

$$Gu_3 = Gu_2 \cup$$
$$\{< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}\}|\mathtt{Z1} = [\mathtt{A1}|\mathtt{Z2}] >,$$
$$< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}\}|\mathtt{A1} = \mathtt{b} >\}$$

$$Gu_4 = Gu_3 \cup$$
$$\{< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}, \mathtt{A1} = \mathtt{a}\}|\mathtt{Y1} = [\mathtt{B1}|\mathtt{Y2}] >,$$
$$< \{\mathtt{X} = [\mathtt{A0}|\mathtt{X1}], \mathtt{A0} = \mathtt{a}, \mathtt{A1} = \mathtt{a}\}|\mathtt{B1} = \mathtt{b} >\}$$

.....

A1 is instantiated to a in the guard part of an element of Gu_4, however, it is instantiated to b in the body part of an element of Gu_3. Thus, $\bigcup_{k=\infty} Gu_k$ is not a guarded stream. Therefor, the synchronized merge of GU_1 and GU_2 cannot be defined.

□

3 Model Theoretic Semantics

This section introduces notions which correspond to the Herbrand base and unit clauses. for parallel logic language based on the notion of guarded streams. First, a parallel language based on Horn logic is presented. The language is essentially a subset of Flat GHC [Ueda 88] with only one system predicate, =: unification of a variable term and a simple term. Furthermore all clauses are assumed to be in a *normal form*, namely all arguments in the head part are different variable terms. However it is not difficult to show that the language presented here does not lose any generality compared to Flat GHC using the modification of the transformation algorithm for the strong normal from [Levi 88].

Let *Pred* be a finite set of predicate symbols. Each element of *Pred* has its arity. We denote each element of *Pred* using lower-case letters.

Def. 15

Let $H, B_1, B_2, \ldots, B_n$ be an atomic formula defined from $Pred$, every term which appears in argument of H be a different variable, and $U_{g1}, \ldots, U_{gm}$ and $U_{b1}, \ldots, U_{bh}$ be simple substitution forms. The following is a *guarded clause.*

$$H : -U_{g1}, \ldots, U_{gm} | U_{b1}, \ldots, U_{bh}, B_1, B_2, \ldots, B_n$$

A finite set of guarded clauses is a *program.*

We define $Var(H) = \{X_1, X_2, \ldots, X_k\}$ when H is $p(X_1, X_2, \ldots, X_k)$.

Def. 16

For a guarded stream GU and an atom $p(X_1, X_2, \ldots, X_k)$,a *pseudo I/O history* t is:

$$p(X_1, X_2, \ldots, X_k) : -GU$$

where $p \in Pred$ with arity k, $X_1, X_2, \ldots, X_k$ are different variables, and for every $gu \in GU$ if $U \in |gu|$ then the left hand side of U is an element of of $V_i(GU)$ for some i where $V_i(GU)$ is defined as follows.

$$\begin{aligned} V_0(GU) &= Var(p(X_1, X_2, \ldots, X_k)) \\ V_{i+1}(GU) &= V_i(GU) \cup \\ &\quad \{X | \exists gu \in GU, \exists uni(Y, \tau) \in |gu|, X \text{ appears in } \tau, \\ &\qquad Y \in V_i(GU) \text{ and } \forall gu' \in GU, \\ &\qquad\quad \text{if } gu' \prec gu \text{ then } X \text{ does not occur in } gu'\} \end{aligned}$$

$p(X_1, X_2, \ldots, X_k)$ is called the *head part of* t and GU is called the *body part of* t. Intuitively, GU only contains variables which are *visible* from outside through the head part.

In pseudo I/O history, the same computation can be represented in several ways. In other words, if t_1 and t_2 are identical except for the names of variables which do not appear in the head parts, they are considered to represent the same computation. Thus an equivalent relation is introduced to the domain of pseudo I/O histories.

Def. 17

A mapping $\sigma : Var \rightarrow Var$ is a *renaming mapping* if there exists a mapping σ', such that

$$\sigma\sigma' = \sigma'\sigma.$$

Let GU be a guarded stream and σ be a renaming mapping. σGU is a guarded stream, defined as follows.

$$\sigma GU = \{\sigma gu | gu \in GU\}$$

where

$$\sigma gu =< \sigma * \theta | uni(\sigma Y, \sigma\tau') >,$$

for $gu =< \theta | uni(Y, \tau') >$ and $\sigma * \theta$ is a substitution defined as follows.

$$\sigma * \theta = \{\sigma X = \sigma\tau | X = \tau \in \theta\}$$

It is easy to show that if GU is a guarded stream, then σGU is also a guarded stream.

Def. 18

Let $t_1 : H : -GU_1$ and $t_2 : H : -GU_2$ be pseudo I/O histories with the same head part H. If there exists a renaming mapping $\sigma : GU_1 \rightarrow GU_2$ invariant on $Var(H)$ such that $\sigma GU_1 = GU_2$ then

$$t_1 \approx t_2$$

holds.

It is easy to show that $\approx$ is a equivalent relation. We denote the quotient set of all pseudo I/O histories with $\approx$ as $I/O - hist$. Each element of $I/O - hist$ is called an *I/O history*.

Def. 19

An *interpretation* is any subset of $I/O - hist$.

Def. 20

Let t be a I/O history and g be a goal. t is a *trace of* g if for each element of t has the form of $H : -GU$ and there exists a ω−substitution σ such that $\sigma H = g$ and the followings are true.

(1) For any $< \theta | U >\in GU$, there exists a set of substitution forms σ' such that $\sigma' \cup \theta = \sigma' \cup \sigma$ and $\sigma' \cup \theta (= \sigma' \cup \sigma)$ is a ω−substitution.

(2) For any $< \theta | U >\in GU$, if U is a substitution from $X = \tau$ then σ does not instantiate X and if U is a test form $X? = \tau$ then σX is equal to $\sigma\tau$.

A mapping σ does not instantiate a variable X if $\sigma X = Y (\in Var)$ and there is no Z such that $\sigma Z = Y$ except X.

Def. 21

Let I be an interpretation and g be a goal. g is *true* on I when there exists a trace of $g \in I$.

For goals $g_1, \ldots, g_n$, let $t_1, \ldots, t_n$ be their traces. If they are I/O histories which are obtained when these goals are executed in parallel, the shared variables in t_i and t_j must have some value, and they occur as subterms of values of the same variables in t_i and t_j. The following notion is introduced to formalize these conditions.

Def. 22

$t_1, \ldots, t_n$ is *variable compatible* if for any i and j $(1 \leq i, j \leq n)$, the set of variables which occur in both t_i and t_j is equivalent to $Common(t_i, t_j)$ defined as follows.

$COM_0(t_i, t_j) = Var(H_i) \cap Var(H_j)$
$COM_{k+1}(t_i, t_j) = COM_k(t_i, t_j) \cup$
$\{X | \exists Y \in COM_k(t_i, t_j), \exists \tau :$ which has the form of $X, [X|Z]$, or $[Z|X]$,
$\exists gu_i \in GU_i, ((Y = \tau) \in |gu_i| \vee (Y? = \tau) \in |gu_i|) \wedge$
$\forall gu_i' \in GU_i$, if $gu_i' \prec gu_i$ then X does not occur in any $U \in |g\ u_i'| \wedge$
$\exists gu_j \in GU_j, ((Y = \tau) \in |gu_j| \vee (Y? = \tau) \in |gu_j|) \wedge$
$\forall gu_j' \in GU_j$, if $gu_j' \prec gu_j$ then X does not occur in any $U \in |g\ u_j'|$.$\}$

$$Common(t_i, t_j) = \bigcup_{k \to \infty} COM_k$$

where H_m denotes the head part of t_m and GU_m denotes the body part. Obviously, for $n = 1$, t_1 is variable compatible.

Def. 23

Let I be an interpretation and $g_1, \ldots, g_n$ be a goal clause. $g_1, \ldots, g_n$ is *true* on I if there exists a trace of $t_i \in I$ for every $i (1 \leq i \leq n)$, and there exists a synchronized merge $GU_1 \| \ldots \| GU_n$ where $GU_1, \ldots, GU_n$ are body parts of elements of $t_1, \ldots, t_n$, which are variable compatible.

The empty goal clause is always true.

It is easy to show by the following proposition that **Def. 22** is well defined, namely the truth value of $g_1, \ldots, g_n$ does not depends on which element is selected from the trace of each goal.

Prop. 1

Let $GU_1, \ldots, GU_n$ be guarded streams and σ be a renaming mapping. If there exists $GU_1 \| \ldots \| GU_n$, then there also exists $\sigma GU_1 \| \ldots \| \sigma GU_n$.

The proof is straightforward from the definition of synchronized merge.

For a given goal g, it is easy to show that g is true if and only if the goal clause with only one goal g is true.

Def. 24

Let GU be a guarded stream and V be a finite set of variables. The *restriction of* GU by V :$GU \downarrow V$ is the set defined as follows.

$$GU \downarrow V = \{< \sigma | uni(X, \tau) > | < \sigma | uni(X, \tau) > \in GU, X \in V_k \text{ for some } k\}$$

where

$$\begin{aligned} &V_0 = V \\ &V_{i+1} = V_i \cup \{X | \exists gu \in GU, \exists uni(Y, \tau) \in |gu|, \\ &\qquad\qquad X \text{ appears in } \tau, Y \in V_i \text{ and } \forall gu' \in GU, \\ &\qquad\qquad \text{if } gu' \prec gu, \text{ then } X \text{ does not occur in } gu\} \end{aligned}$$

If GU is a guarded stream then $GU \downarrow V$ is also a guarded stream.

Def. 25

Let GU be a guarded stream and θ be a set of simple substitution form. The set $GU \bowtie \theta$ is defined as follows if it is a guarded stream.

$$GU \bowtie \theta = \{< \sigma | U_h > | < \sigma' | U_h > \in GU, \sigma = \theta \cup \sigma'\}$$

Def. 26

Let D be a finite set of guarded clauses and I be an interpretation. *I is a model of D* if for any $t \in I$, there exists a clause $H : -U_{g1}, \ldots, U_{gm} | X_1 = \tau_1, \ldots, X_k = \tau_k, B_1, \ldots, B_k \in D$, each element of t has the following form:

$$\begin{aligned} H : -\{&< \{U_{g1}, \ldots, U_{gm}, \} | uni(X_1, \tau_1) >, \ldots, \\ &< \{U_{g1}, \ldots, U_{gm}, \} | uni(X_k, \tau_k) > \} \cup \\ &((GU_1 \| \ldots \| GU_k) \bowtie \{U_{g1}, \ldots, U_{gm}\}) \downarrow Var(H) \end{aligned}$$

where GU_i is the body part of some instance of a tracc $t_i (\in I)$ of the goal σB_i for some $\omega-$substitution $\sigma = \{U_{g1}, \ldots, U_{gm}\} \cup \{X_1 = \tau_1, \ldots, X_k = \tau_k\} \cup \sigma'$, which is restricted to $Var(H) \cup \{X_1, \ldots, X_n\}$ and

$$\forall < \theta | U > \in (GU_1 \| \ldots \| GU_k) \bowtie \{U_{g1}, \ldots, U_{gm}\}, \theta \subset \sigma.$$

The following proposition is easy to show from the definition of models.

Prop. 2

Let $M_i (i \in Ind)$ be a class of models of D for a set of indices Ind. Then,

$$\bigcup_{i \in Ind} M_i$$

is also a model of D.

From **Prop. 2**, it is easy to show that there exists a unique maximum model for a given D. The semantics of D is defined as the maximum model of D. A goal clause $g_1, \ldots, g_m$ is true on D if it is true on the maximum model of D. Intuitively, the maximum model is the set of all computations without failure or deadlock on D.

Example 3

For the following program D:

```
shuffle(X, Y, Z) :- X = [A1|X1] |
                    Z = [A1|Z1], shuffle(X1, Y, Z1).
shuffle(X, Y, Z) :- Y = [B1|Y1] |
                    Z = [B1|Z1], shuffle(X, Y1, Z1).

inva(Z, Y) :- Z = [A|Z1], A = a |
              Y = [B|Y1], B = b, inva(Z1, Y1).
inva(Z, Y) :- Z = [B|Z1], B = b | inva(Z1, Y).
```

The maximum model of D contains the elements of the form such as:

$$\texttt{inva(Z,Y)} : -GU_1,$$
$$\texttt{shuffle(X,Y,Z)} : -GU_2$$

where GU_1 and GU_2 are introduced in **Example 1**. For goals of the form `inva(Z, Y)` and `shuffle(X, Y, Z)`, these goals can run as GU_1 and GU_2 respectively when they are executed independently. However it can be shown that if these goals are executed in parallel as the goal clause `inva(Z,`

Y), shuffle(X, Y, Z), then they do not run as GU_1 or GU_2 because the synchronized merge of GU_1 and GU_2 cannot be defined.

In general, if goal clause inva(Z, Y), shuffle(X, Y, Z) is executed with an infinite number of "a"s in X as the input, the number of "b"s in the instantiated part of the output Z never exceeds the number of "a"s in Z. It can be discussed by showing that for any result of synchronized merge of traces of each goal, the number of "b"s in the output does not exceed the number of "a"s in the output on this semantics. This kind of discussion is impossible in the method of [Levi 88] or on semantics for pure Horn logic programs with a complete Herbrand universe such as [Sakakibara 85].

□

The semantics presented here is defined for characterizing the goal clauses which runs *normally* without failure or deadlock on the program as true on the model of the program. However *goal clauses which run normally*are not necessarily the successful goals. That is goals which run infinitely are regarded as goals that run normally. Furthermore a goal clause suspending goal can also be true. Consider the following example:

```
p(X, Y) :- X = a | Y = b.
q(X, Y) :- Y = b | X = a.
t(X, Y) :- X = a | true, p(X, Y), q(X, Y).
```

Although goal t(X, Y) suspends on this program, if X is instantiated to a, then the execution proceeds. The maximum fixpoint of this program contains:

$$t(X,Y) :- \{< \{X = a\}|true >, < \{X = a\}|Y = b >, < \{X = a\}|X? = a >\}$$

Thus, goal t(X, Y) is true on this program. On the other hand, let us consider the program obtained from the previous program by replacing the third clause with:

```
t :- true | true, p(X, Y), q(X, Y).
```

In this case, when goal `t` is invoked then goal clause `p(X, Y), q(X, Y)` is invoked and suspends, it cannot proceeds the computation whatever process runs in parallel with `t`. Goal such as `t` are false on the semantics presented here.

4 Fixpoint Semantics

This section discusses the fixpoint characterization of the semantics of programs.

It is easy to show that the set of all interpretations IP defined from $I/O - hist$ is a complete lattice with a partial order of set inclusion. The maximum element is $I/O - hist$ and the minimum element is ϕ.

Def. 27

Let D be a program. $\Phi_D : IP \rightarrow IP$ is the function defined as follows.

$$\Phi_D(S) = S \cap$$
$$\{t | \text{ each element of } t \text{ has the form of}$$
$$H :- \{< \{U_{g1}, \ldots, U_{gm}\} | uni(X_1, \tau_1) >, \ldots,$$
$$< \{U_{g1}, \ldots, U_{gm}\} | uni(X_h, \tau_h) >\} \cup$$
$$((GU_1 \| \ldots \| GU_k) \bowtie \{U_{g1}, \ldots, U_{gm}\}) \downarrow Var(H)$$

for some clause

$$H :- U_{g1}, \ldots, U_{gm} | X_1 = \tau_1, \ldots, X_h = \tau_h, B_1, \ldots, B_k \in D$$

where GU_i is the body part of a element of
the trace of $\sigma B_i . \sigma$ is a $\omega -$ substitution such that

$$\sigma = \{U_{g1}, \ldots, U_{gm}, X_1 = \tau_1, \ldots, X_h = \tau_h\} \cup \sigma'$$

for some σ', restricted to $Var(H) \cup \{X_1, \ldots, X_h\}$, and

$$\theta \subset \sigma$$

for all $< \theta | U > \in (GU_1 \| \ldots \| GU_k) \bowtie \{U_{g1}, \ldots, U_{gm}\}.\}$

For a chain $S_i : S_1 \supset S_2 \supset \ldots$, the greatest lower bound of S_i is denoted $\bigcap S_i$.

Def. 28

Let L be a complete lattice. A function $f : L \rightarrow L$ is *$\omega-$continuous from below*, if for any chain$S_i : S_1 \supset S_2 \supset \ldots$,

$$\bigcap\{f(S_i)|0 \leq i\} = f(\bigcap\{S_i|0 \leq i\}).$$

It is well known that if f is $\omega-$continuous from below then f is monotone, that is if $S_1 \supset S_2$, then$f(S_1) \supset f(S_2)$. The following two propositions are well known (see [Park 69]).

Prop. 3

Let L be a complete lattice with the maximum element, $\top$. If $f : L \rightarrow L$ is $\omega-$continuous from below, then f has the greatest fixpoint $\mathsf{gfp}f$ and,

$$\mathsf{gfp}f = \bigcap\{f^n(\top)|n \geq 0\}$$

where $f^0(X) = X, f^{n+1}(X) = f(f^n(X))$.

Prop. 4

If f is a monotone function, then:

$$\mathsf{gfp}f = \bigcup\{X|f(X) \supset X\}$$

where $\bigcup S$ is the least upper bound of S.

The following properties are easy to show from the definitions.

Prop. 5

Φ_D is $\omega-$continuous from below.

proof:

(1) $\Phi_D(\bigcap S_j) \subset \bigcap\{\Phi_D(S_j)\}$:

For any $t \in \Phi_D(\bigcap S_j)$, there exists a clause:

$$H : -U_{g1}, \ldots, U_{gm}|X_1 = \tau_1, \ldots, X_h = \tau_h, B_1, \ldots, B_k \in D$$

An element of t has the form as following.

$$H : -\{< \{U_{g1}, \ldots, U_{gm}\}|uni(X_1, \tau_1) >, \ldots,$$
$$< \{U_{g1}, \ldots, U_{gm}\}|uni(X_h, \tau_h) >\}\cup$$

$$((GU_1 \| \ldots \| GU_k) \bowtie \{U_{g1}, \ldots, U_{gm}\}) \downarrow Var(H)$$

where GU_i is the body part of an element of a trace $t_i (\in \bigcap\{S_j\})$ of σB_i and σ is a ω−substitution which satisfies the condition in **Def. 27**. From $t_i \in \bigcap\{S_j\}$, $t_i \in S_j$ for all j. Thus for all j, $t \in \Phi_D(S_j)$. Since, $\bigcap\{\Phi_D(S_j)\}$ is the greatest lower bound of $\{\Phi(S_i)\}$, then:

$$t \in \bigcap\{\Phi_D(S_j)\}.$$

(2) $\Phi_D(\bigcap S_j) \supset \bigcap\{\Phi_D(S_j)\}$:

Assume $t \in \bigcap\{\Phi_D(S_j)\}$. For any j, $t \in \Phi_D(S_j)$. An element of t has the form of

$$\begin{aligned} H : -\{ &< \{U_{g1}, \ldots, U_{gm}\} | uni(X_1, \tau_1) >, \ldots, \\ &< \{U_{g1}, \ldots, U_{gm}\} | uni(X_h, \tau_h) > \} \cup \\ &((GU_1 \| \ldots \| GU_k) \bowtie \{U_{g1}, \ldots, U_{gm}\}) \downarrow Var(H) \end{aligned}$$

where GU_i is the body part of an element of a trace $t_i (\in \bigcap\{S_j\})$ of σB_i and σ is a ω−substitution which satisfies the condition in **Def. 27**. For any j, since $t_i \in S_j$, then $t_i \in \bigcap\{S_j\}$ for each i. Thus, $t \in \Phi_D(\bigcap\{S_j\})$.

□

Prop. 6

$\Phi_D(I) \supset I$ if and only if I is a model of D.

proof:

if part:

Let I be a model of D and $t \in I$. From **Def. 26**, there exists a clause:

$$H : -U_{g1}, \ldots, U_{gm} | X_1 = \tau_1, \ldots, X_h = \tau_h, B_1, \ldots, B_k \in D$$

and t has the form as following.

$$\begin{aligned} H : -\{ &< \{U_{g1}, \ldots, U_{gm}\} | uni(X_1, \tau_1) >, \ldots, \\ &< \{U_{g1}, \ldots, U_{gm}\} | uni(X_h, \tau_h) > \} \cup \\ &((GU_1 \| \ldots \| GU_k) \bowtie \{U_{g1}, \ldots, U_{gm}\}) \downarrow Var(H) \end{aligned}$$

where GU_i is the body part of an element of a trace $t_i \in I$ of σB_i and σ is a $\omega-$substitution which satisfies the condition in **Def. 27** . From the definition of Φ_D, $t \in \Phi_D(I)$.

only if part:

Assume $\Phi_D(I) \supset I$, in other words for any $t \in I$, $t \in \Phi_D(I)$. From the definition of $\Phi_D(I)$,

$t \in \{t|$ there exists a clause

$$H : -U_{g1}, \ldots, U_{gm} | X_1 = \tau_1, \ldots, X_h = \tau_h, B_1, \ldots, B_k \in D,$$

each element of t has the following form:

$$H : -\{< \{U_{g1}, \ldots, U_{gm},\} | uni(X_1, \tau_1) >, \ldots,$$
$$< \{U_{g1}, \ldots, U_{gm},\} | uni(X_h, \tau_h) > \} \cup$$
$$((GU_1 \| \ldots \| GU_k) \bowtie \{U_{g1}, \ldots, U_{gm}\}) \downarrow Var(H)$$

where GU_i is the body part of some instance of a trace $t_i (\in I)$ of the goal σB_i, σ is a $\omega-$substitution such that

$$\sigma = \{U_{g1}, \ldots, U_{gm}\} \cup \{X_1 = \tau_1, \ldots, X_h = \tau_h\} \cup \sigma',$$

which is restricted to $Var(H) \cup \{X_1, \ldots, X_n\}$ and

$$\forall < \theta | U > \in (GU_1 \| \ldots \| GU_k) \bowtie \{U_{g1}, \ldots, U_{gm}\}, \theta \subset \sigma.\}$$

This is the definition of the model of D.

□

Thus, we derive the following theorem.

Theorem

Let D be a program and M_D be the maximum model of D. Then:

$$M_D = \bigcap \{\Phi_D^n(I/O - hist) | n \geq 0\}.$$

5 Conclusion

This paper presented a new declarative semantics for a subset of Flat GHC programs based on the maximum model. Using the semantics, the solutions

of programs which contain perpetual processes controlled by guard commit mechanisms can be characterized as the logical consequence of the programs.

The semantics presented here is a kind of success set semantics. Thus, it is enough to discuss the results of normal computation. However, a true goal clause on this semantics is a goal clause which can run normally. This does not mean that a true goal clause always runs normally. For example, if a subgoal of the clause commits to a different clause, then it can fail or deadlock. GHC is a *don't care non-deterministic language*. Thus, a method to characterize the set of goal clauses which run in any case is also expected. Such a set is characterized as the set of goal clauses which are not in the set of clauses that can fail or deadlock. A result of such a *failure set* is reported in [Falaschi 88] for a sequential logic language. The author believe that it is possible to characterize the failure set for parallel logic languages using notions presented here, such as guarded stream and synchronized merge.

6 Acknowledgments

I would like to thank Dr. Furukawa, the researchers of the First Laboratory of ICOT and Professor Levi of Università di Pisa for their helpful discussions.

References

[Apt 82] K. Apt and M. H. Van Emden, Contributions to the theory of logic programming, J. Assoc. Comput. Mach. 29, (1982)

[Brock 81] J. D. Brock and W. B. Ackermann, Scenarios: A Model of Non-determinate Computation, Lecture Notes in Computer Science, No. 107, Springer, 1981

[Clark 86] K. L. Clark and S. Gregory, PARLOG: Parallel programming in logic, ACM Trans. on Programming Language and Systems 86, 1986

[Falaschi 88a] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi, A more general Declarative Semantics for Logic Programming

Languages, Dipartimento di Informatica, Università di Pisa, Italy, Tech. Report, January 1988

[Falaschi 88b] M. Falaschi and G. Levi, Operational and fixpoint semantics of a class of commited-choice logic languages, Dipartimento di Informatica, Università di Pisa, Italy, Techn. Report, January 1988

[Levi 87] G. Levi and C. Palamidessi, An Approach to the Declarative Semantics of Synchronization in Logic Languages, Proc. of International Conf. on Logic Programming 87, 1987

[Levi 88] G. Levi, A new declarative semantics of Flat Guarded Horn Clauses, ICOT Technical Report, 1988

[Lloyd 84] J. W. Lloyd, Foundations of logic programming, Springer-Verlag, 1984

[Maher 87] M. J. Maher, Logic Semantics for a Class of Commited-Choice Programs, Proc. of International Conf. on Logic Programming 87, 1987

[Park 69] D. Park, Fixpoint induction and proofs of program properties, Machine Intelligence 5, Edinburgh University Press, Edinburgh, 1969

[Sakakibara 85] Y. Sakakibara, A Fixpoint Characterization of Stream Parallelism in Logic Programs, Proc. of 2nd Cof. JSSST, 1985 (in Japanese)

[Saraswat 85] V. A. Saraswat, Partial Correctness Semantics for CP[↓, |, &], Lecture Notes in Comp. Sci., No. 206, 1985

[Saraswat 87] V. A. Saraswat, The Concurrent logic programming CP: definition and operational semantics, Proc. of ACM Symp. on Principles of Programming Languages, 1987

[Shapiro 86] E. Y. Shapiro, Concurrent Prolog: A progress report, Lecture Notes in Comp. Sci. No. 232, 1986

[Shibayama 87] E. Shibayama, A Compositional Semantics of GHC, Proc. of 4th Cof. JSSST, 1987

[Takeuchi 86] A. Takeuchi, Towards a Semantic Model of GHC, Tech. Rep. of IECE, COMP86-59, 1986

[Ueda 88] K. Ueda, Guarded Horn Clauses, MIT Press, 1988

[Ueda 86] K. Ueda, On Operational Semantics of Guarded Horn Clauses, ICOT Technical Memo, TM-0160, 1986