ICOT Technical Memorandum: TM-0602

TM-0602

A Declarative Semantics of Parallel Logic Programs based on Failure Deadlock Set

by
M. Murakami

October, 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 based on Failure/Deadlock Set

Masaki Murakami [1]

Extended Abstract

September 18, 1988

Abstract

This paper presents a declarative semantics of Flat GHC programs. The semantics presented here is based on the failure/deadlock set of a Flat GHC program, namely the set of the I/O histories representing computations which fail or fall into deadlock within finite steps.

1 Introduction

We reported the success set semantics of Flat GHC programs [Murakami 88]. The semantics presented in the paper is an extension or a modification of the model theoretic semantics of pure Horn logic programs [Apt 82, Lloyd 84], namely the semantics of a program is a model of the set of formulas that define the program. We defined that a goal clause is true on the model if and only if the goal clause is $\omega-$ successful (the goal clause can be executed without deadlock or failure). The domain of I/O history is introduced instead of the standard Harbrand universe. The denotation of a program, the $\omega-$ success set is defined as a set of I/O histories. Using the semantics, the solutions of programs that contain perpetual processes controlled by guard commit mechanisms can be characterized as the logical consequence of the programs.

However, in the case of programs of committed choice language such as GHC, the set of successful goals and the set of goals which have possibility to fall into deadlock or failure can have non-empty intersection. Thus it is impossible to discuss whether a goal clause can fall into deadlock or fail with success set only [Falaschi 88]. Thus [Falaschi 88] reported a new approach to give a semantics to committed choice logic languages. In that paper, the semantics of program is defined as a tuple of the success set and the failure set. A goal clause is *true* if it can be fail in that approach. However in that paper, the model is defined as a set of formulas which contain only the information of the final results which are obtained when the computation is terminated. Thus non terminating computations cannot be discussed in that approach. For instance, a goal clause g_1, g_2 is not true if g_1 does not terminate and g_2 fails.

In this paper, we define the failure/deadlock set, the set of goals which can fall into deadlock or finite failure. Namely, we define that a goal clause is true on the model of a program when on of the goal clause has possibility of deadlock or finite failure with the program. The set of failure/deadlock set of a program is defined as the least model of the program.

[1]Institute for New Generation Computer Technology,
Mita Kokusai Building, 21F, 4-28, Mita 1-chome, Minato-ku, Tokyo 108 Japan, Phone +81-03-456-2514
e-mail: murakami@icot.junet

Thus the semantics of programs defined as a tuple of the $\omega-$ success set and failure/deadlock set as [Falaschi 88]. Existence of processes which have possibility of failure/deadlock can be discussed for the programs with perpetual processes.

2 Guarded Streams

In this section, we define the notion of the *guarded streams*. Guarded streams are introduced in [Murakami 88] first. However, in that paper, only computations without failure/deadlock are represented with guarded streams. A new definition of guarded streams is presented in order to discuss computations with failure/deadlock.

Let *Var* be an enumerable set of variables, *Fun* be a set of function symbols. Each element of *Fun* has its arity. Let *Terms* be the set of terms defined from *Fun* and *Var* in standerd way. A term τ is *simple* if it is a 0-ary function symbol or the form of $f(X_1, \ldots, X_n)$ where $f \in Fun$ and $X_1, \ldots, X_n$ are different variables. *Substitutions* on *Terms* are defined as usual.

In this paper, we consider programs on the domain of lists of { `a, b` } as examples, thus $\mathtt{a}, \mathtt{b}, \mathtt{cons} \in Fun$. The arity of `a`, `b` and `nil` is 0, and the arity of `cons` is 2. $\mathtt{cons}(X, Y)$ is denoted $[X|Y]$ and `nil` is denoted [ ] as usual.

Def. 1

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,

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

Def. 2

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

The notion of I/O history introduced in this paper corresponds to the notion of element of the Herbrand base 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:

$$H : -GU.$$

GU is a set of tuples $< \sigma | U_b >$ where σ is a substitution which is requierd to solve the guards that appear before committing some clause and U_b is a expression which express an execution of unification in the body part of the clause to which the goal committed. Intuitively, $< \sigma | U_b >$ means that if the arguments of the process are instantiated with σ then unification U_b can be 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`.

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

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 scheduling.

In this paper, we informally define that a computation fails or falls into deadlock when a goal commits to some clause such that there is no $(\omega-)$ successful computation after the commit. A computation which fails or falls into deadlock is represented by a guarded stream which contains $\perp$ instead of U_b.

For instance, consider the following program:

```
p(X, Y) :- X = a | Y = [b, Z], q1(Z).
p(X, Y) :- X = a | Y = [a, Z], q2(Z).

q1(Z) :- true | Z = b, r(W).
q2(Z) :- true | Z = b.

r(W) :- W = a | true.
```

For this program, a goal `p(a, Y)` cannot avoid deadlock if it commits to the first clause. This situation is represented with the following guarded stream.

$$\{< \{\texttt{X} = \texttt{a}\} | \ >\}$$

On the other hand, if it commits to the second clause, the computation continues. This situation is represented with the following guarded stream.

$$\{< \{\texttt{X} = \texttt{a}\} | \texttt{Y} = [\texttt{a}, \texttt{Z}] >, < \{\texttt{X} = \texttt{a}\} | \texttt{Z} = \texttt{b} >\}$$

Def. 3

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

$$X? = \tau$$

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

Def. 4

Let σ be a substitution and $uni(X,\tau)$ be a substitution form $X = \tau$ or a test form $X? = \tau$ for $X \in Var$ and a simple term τ. $< \sigma|U >$ is a *guarded unification* where U is $uni(X,\tau)$ or $\bot$. σ is the *guard part* of $< \sigma|U >$ and U 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. If the active part is $\bot$, it means that a goal such that failure/deadlock is unavoidable is invoked.

Def. 5

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

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

if U is a test form or a substitution form, and

$$| < \sigma|U > | = \sigma$$

if $U = \bot$.

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}.

Def. 6

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. 7

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

1) 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 U_1 or U_2 is a test form and their right hand sides are unifiable. 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.

2) If $< \sigma|U > \in GU$, then $(X = \tau) \notin \sigma$ for any $< \theta|X = \tau > \in GU$.

3) For any $< \theta|X? = \tau > \in GU$, if τ and τ' are not unifiable, then $(X = \tau') \notin \sigma$ for $< \sigma|U > \in GU$.

4) 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 τ and τ' are unifiable.

Conditions 1),...,4) mean that all variable in GHC programs are logical variables and if they are instantiated, the values are never changed.

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.

U denotes a substitution form, test form or $\bot$.

Def. 8

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

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

$$\begin{aligned} Gu_{k+1} = Gu_k \cup \{< \sigma | U > | & \exists i, \exists < \sigma' | U > \in GU_i, \forall (X = \tau) \in \sigma', \\ & ((\forall j, < \sigma'' | X = \tau > \notin GU_j) \vee < \sigma'' | X = \tau > \in Gu_k) \wedge \\ & \sigma = (\sigma' - \{X = \tau | < \sigma'' | X = \tau > \in Gu_k\}) \cup \\ & \quad \{U | U \in \sigma'' < \sigma'' | X = \tau > \subset Gu_k\}\} \end{aligned}$$

and let GU be as follows.

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

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.

Def. 9

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

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

Def. 10

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

$$\begin{aligned} & V_0 = V \\ & V_{i+1} = V_i \cup \{X | \exists gu \in GU, \exists uni(Y, \tau) \in |gu|, \\ & \qquad X \text{ appears in } \tau, Y \in V_i \text{ and } \forall gu' \in GU, \end{aligned}$$

if $gu' \prec gu$, then X does not occur in $gu\}$

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

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]. We denote set of predicate symbols as *Pred*.

Def. 11

Let $H, B_1, B_2, \ldots, B_n$ be atomic formulas constructed with *Terms* and *Pred* where all arguments of H are different variables, and $U_{g1}, \ldots, U_{gm}, U_{b1}, \ldots, U_{bh}$ be simple substitution forms. The following formula is a *guarded clause*.

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

A program is a finite set of guarded clauses.

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

Def. 12

Let p be an element of *Pred* with arity k, $X_1, X_2, \ldots, X_k$ be different variables and σ be an $\omega-$ substitution. Then $\sigma p(X_1, X_2, \ldots, X_k)$ is a *goal*.

Def. 13

A sequence of goals: $g_1, \ldots, g_n$ is a *goal clause*.

Def. 14

For a guarded stream GU and an atom $p(X_1, X_2, \ldots, X_k)$,a *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.

$$V_0(GU) = Var(p(X_1, X_2, \ldots, X_k))$$

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

$p(X_1, X_2, \ldots, X_k)$ 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.

The concept of I/O histories corresponds to the concept of unit clauses of the standard model theoretic semantics of pure Horn logic programs. However in 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 the equivalent relation based on renaming of variables should be introduced. In the following, we denote the set of representatives of equivalence classes of all I/O histories defined from *Fun*, *Var* and *Pred* as $I/Ohist$.

Def. 15

Let $H : -GU$ be an I/O history. If U is a substitution form or a test form for all $< \sigma|U > \in GU$, then $H : -GU$ is a *successful history*. If there is a $< \sigma|U >$ such that U is $\perp$, then $H : -GU$ is a *unsuccessful history*.

$I/Ohist$ is divided into two disjoint subsets, $I/Ohist_\infty$: the set of all successful histories and $I/Ohist_\perp$: the set of all unsuccessful histories.

Def. 16

Any subset of $I/Ohisi_\infty$ is an ∞ *interpretation*. Any subset of $I/Ohist_\perp$ is a $\perp$ *interpretation*.

Def. 17

Let t be an I/O history and g be a goal. $H : -GU$ is a *trace of* g if the following (1),...,(3) hold.

(1) There exists an $\omega-$ substitution σ such that $\sigma H = g$.

(2) For any $< \theta|U > \in GU$, $\theta \subset \sigma$.

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

σ does not instantiate a variable X if $\sigma X = Y (\in Var)$ and there does not exist Z such that $\sigma Z = Y$ except X.

Let t be a trace of a goal g. If t is a successful history, it is a *successful trace of* g. If t is a unsuccessful history, it is a *unsuccessful trace of* g.

Def. 18

Let $I_\perp$ be a $\perp$ interpretation and g be a goal. g is *true* on $I_\perp$ if there exists an $\omega-$ substitution, σ and there exists an unsuccessful trace of $\sigma g \in I_\perp$. g is *true* on a successful interpretation I_∞ when there exists a successful trace of $\sigma g \in I_\infty$ for some $\omega-$substitution: σ.

Def. 19

Let I_∞ be a successful 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 $t_i \in I_\infty$ for every $\sigma g_i (1 \leq i \leq n)$ for some $\omega-$substitution: σ, and there exists a synchronized merge $GU_1 \| \ldots \| GU_n$ where $GU_1, \ldots, GU_n$ are body parts of $t_1, \ldots, t_n$.

The empty goal clause is always true.

Def. 20

Let $I_\perp$ be a $\perp$ interpretation and I_∞ be a ∞ interpretation. I_∞ and $I_\perp$ is *consistent* if for any θ such that $\sigma \subset \theta$, if I_∞ dose not contain any trace of θH then

$$H : - < \sigma| \perp > \in I_\perp$$

Let $\langle I_\perp, I_\infty \rangle$ be a consistent tuple of $\perp$ interpretation $I_\perp$ and I_∞ interpretation I_∞

Def. 21

A goal clause $g_1, \ldots, g_n$ is true if there exists an $\omega-$ substitution: σ such that for each $g_i (1 \leq i \leq n)$, a trace of σg_i: t_i is in $I_\perp \cup I_\infty$, there exists $j (1 \leq j \leq n)$ such that $t_j \in I_\perp$ and there exists a synchronized merge $GU_1 \| \ldots \| GU_n$ where $GU_1, \ldots, GU_n$ are body parts of $t_1, \ldots, t_n$.

The empty goal clause is always true.

Def. 22

A guarded clause:

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

is true on $\langle I_\perp, I_\infty \rangle$ if the following condition is true.

If there exists an $\omega-$ substitution: σ which does not instantiate variables which are invisible from outside through H and makes $B_1, \ldots, B_k$ true and GU is a guarded stream then :

$$H : -GU \in I_\perp,$$

where GU is a set of guarded unifications such as:

$$GU = \{< \{U_{g1}, \ldots, U_{gm},\} | U_{b1} >, \ldots, < \{U_{g1}, \ldots, U_{gm},\} | U_{bh} >\} \cup ((GU_1 \| \ldots \| GU_k) \bowtie \{U_{g1}, \ldots, U_{gm}\}) \downarrow Var(H)$$

and GU_i is a body part of a trace ($\in I_\perp \cup I_\infty$) of a goal σB_i.

$\omega-$ success set of program D is the maximum model of D defined in [Murakami 88].

Def. 23

Let D be a GHC program, M_∞^D be the $\omega-$ success set of D. $\perp$ interpretation: $I_\perp$ is a *$\perp$ model of D* if following conditions are true.

(1) $I_\perp$ and M_∞^D is consistent.

(2) All clause in D is true on $\langle I_\perp, M_\infty^D \rangle$.

(3) Let $H : -U_{g1}, \ldots, U_{gm} | U_{b1}, \ldots, U_{bh}, B_1, \ldots, B_k \in D$, and σ be a $\omega-$ substitution. For any $\omega-$ substitution: θ such that $\sigma \subset \theta$ and θ does not instantiate variables which are invisible from H, if $\theta' B_1, \ldots, \theta' B_k$ is not true on M_∞^D where $\theta' = \theta \cup \{U_{g1}, \ldots, U_{gm}\} \cup \{U_{b1}, \ldots, U_{bh}\}$ then

$$H : \quad < \sigma \cup \{U_{g1}, \ldots, U_{gm}\} \cup \{U_{b1}, \ldots, U_{bh}\} | \perp > \in I_\perp.$$

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

Prop. 1

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

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

is also a $\perp$ model of D.

From **Prop. 1**, it is easy to show that there exists a unique least $\perp$ model for a given D. The least $\perp$ model of D is the *failure/deadlock set* of D and denoted as $M_\perp^D$. The semantics of D is defined with $\langle M_\perp^D, M_\infty^D \rangle$.

4 Conclusion: Relation to the Operational Semantics

This paper presented a new model theoretic semantics for Flat GHC programs based on $\omega-$ success set and failure/deadlock set.

We defined the notion of true for goal clauses and sets of guarded clauses to characterize failure/deadlock of programs. We denote failure/deadlock with the symbol $\perp$. Note that $\perp$ does not mean that failure/deadlock has happened at this moment. $\perp$ means that a goal made a commit which makes deadlock/fail unavoidable. In other words a goal clause is true on $\langle M_\perp^D, M_\infty^D \rangle$ if and only if a subgoal g can be spawned which makes a commit such that any instantiation to the arguments of g cannot keep the computation from failure/deadlock any longer. Actually fail/deadlock will happen within finite steps after the commit.

Yet another model theoretic characterization of failure/deadlock may be possible. We expect further discussion on the characterization of failure/deadlock. A fixedpoint characterization of the failure/deadlock set is also expected in the future.

5 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)

[Falaschi 88] M. Falaschi and G. Levi, Operational and fixpoint semantics of a class of committed-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 Committed-Choice Programs, Proc. of International Conf. on Logic Programming 87, 1987

[Murakami 88] M. Murakami, A New Declarative Semantics of Parallel Logic Programs with Perpetual Processes, to appear in Int. Conf.on Fifth Generation Computer System 1988, 1988

[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

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

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

[Ueda 88] K. Ueda, Transformation Rules for GHC Programs, to appear in Int. Conf.on Fifth Generation Computer System 1988, 1988