TR-442

A Failure Set Semantics of Guarded Horn Clauses Programs

by

M. Murakami

December, 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 Failure Set Semantics of Guarded Horn Clauses Programs

Masaki Murakami

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

November 25, 1988

Abstract

A success set semantics of GHC programs was reported in [Murakami 88]. That paper adopted the approach which is an extension of declarative semantics of pure Horn logic programs [Lloyd 84]. The semantics of a program is defined as a set of I/O histories instead of unit clauses. This paper presents a failure set semantics of programs as the set of I/O histories. The semantics of programs is defined as the tuple of the $\omega-$success set and the failure set. It also shows that the failure set of a program is characterized as the least fixpoint of a continuous function.

1 Introduction

We reported a declarative semantics of Flat GHC programs [Murakami 88]. The semantics presented in that paper is an extension of the model theoretic semantics of pure Horn logic programs [Apt 82, Lloyd 84]. 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 Herbrand universe. The denotation of a program, the $\omega-$ success set is defined as a set of I/O histories. However, in the case of programs of a committed choice language such as GHC, the set of successful goals and the set of goals which can fall into deadlock or failure can have a non-empty intersection. Thus, it is impossible to discuss whether a goal clause can fall into deadlock or fail only with the success set [Falaschi 88]. [Falaschi 88] reported a new approach to give a semantics to committed choice logic languages. In that paper, the semantics of a program is defined as a tuple of the success set and the failure set. A goal clause is *true* if it can 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.

We consider that languages such as GHC are designed for writing programs where a number of perpetual processes are invoked and there are communications with outside during the execution [Ueda 89]. Thus, a formal semantics should be designed to formalize such infinite computation. We reported a declarative semantics based on the $\omega-$success set [Murakami 88].

In this paper, we present a failure set semantics like [Falaschi 88] for programs which contains perpetual processes based on the notion of I/O histories. In this approach, the notion of truth of a goal clause corresponds to the notion of failure of unification in the body part of a clause. The failure set of a program is defined as the minimum model of the program. The semantics of a program is defined with the tuple of the $\omega-$success set and the failure set as [Falaschi 88]. Thus, the properties of programs which contain perpetual computation can be discussed using the semantics. We also show that the semantics is characterized as the least fixpoint of the function obtained from the program.

2 Guarded Stream

The notion of I/O history introduced in this paper corresponds to the notion of the 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 the head part H, which denotes a form of a process, and the body part CU, which denotes a trace of inputs and outputs of the process:

$$H : -GU.$$

H has the form of $p(X_1, \ldots, X_k)$ where p is a predicate symbol and $X_1, \ldots, X_k$ are different variables. GU is a set of tuples $< \sigma | U_b >$ where σ is a substitution and U_b is an expression which expresses an execution of unification in the body part of the clause. Intuitively, $< \sigma | U_b >$ means that the arguments of the process are instantiated with σ, then unification U_b is executed.

Guarded streams were first introduced in [Murakami 88]. 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.

2.1 Basic Notions

Let *Var* be an enumerable set of variables, and *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 a standard way. A term τ is *simple* if it is a 0-ary function symbol or in 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.

Definition 1

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

$$X = \tau$$

is a *substitution form*. $X = X$ is denoted *true*.

A substitution σ is denoted using a finite set of simple substitution forms.

Definition 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 ω*-substitution*.

$$\begin{aligned}
&\theta_0 = \theta \\
&\theta_{k+1} = \theta_k \cup \\
&\qquad \{X = \tau | X \text{ occurs in } \tau' \text{ for some } (Y = \tau') \in \theta_k,
\end{aligned}$$

$(X = \tau'') \notin \theta_k$ and no variables occurring in τ
occur in the left part of $\theta_k\}$

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

First, we informally present how a guarded stream represents a computation of GHC program using examples. The formal definition is given in the next section.

The following is an example of a normal computation which does not fail. For 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 an 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.

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

Then we show an example that represents a computation with failure. In this paper, a computation with failure is represented with a guarded steam which contains $\perp$ instead of U_b. For instance, consider the following programs.

```
    p(X, Y) :- X = b | Y = a
```

If a goal p(X, Y) receives an input substitution $\{\mathtt{X} = \mathtt{b}, \mathtt{Y} = \mathtt{b}\}$, then it can commit to the clause and fail. Such execution is represented with the following guarded stream.

$$\{< \{\mathtt{X} = \mathtt{b}, \mathtt{Y} = \mathtt{b}\}| \perp >\}$$

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.

2.2 Formal Definitions

Definition 3

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

$$X? = \tau$$

is a *test form*.

Definition 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 $\perp$. 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 $\perp$, it means that an unsuccessful unification is invoked.

Definition 5

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

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

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

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

if $U = \perp$.

Let GU be a set of guarded unifications which represents a computation of a program. Then, GU is well founded with the partial order of execution, that is, 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}.

Definition 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.

Definition 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 the 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) to 4) mean that all variables 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 $\perp$.

Definition 8
Let $GU_1, \ldots, GU_n$ be guarded streams, and $Gu_k (1 \leq k)$ be as follows.

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

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

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.

Definition 9

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.

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

where

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

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

Definition 10

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.

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

3 Model Theoretic Semantics

This section introduces the notion of the model of programs 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 89] 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*, that is, 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 to the strong normal form [Levi 88]. We denote set of predicate symbols as *Pred*.

Definition 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 let $U_{g1}, \ldots, U_{gm}$, $U_{b1}, \ldots, U_{bh}$ 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 $p(X_1, X_2, \ldots, X_k)$.

Definition 12

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

Definition 13

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

Definition 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

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

$p(X_1, X_2, \ldots, X_k)$ 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 $I/Ohist$.

Definition 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 an *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.

Definition 16
Any subset of $I/Ohist_\infty$ is an ∞ *interpretation.* Any subset of $I/Ohist_\perp$ is a $\perp$ *interpretation.*

Definition 17
Let t be an I/O history and g be a goal. $H : -GU$ is a *trace of* g if following (1) to (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 $\sigma X = Y (Y \in Var)$ and there is no 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 an unsuccessful history, it is an *unsuccessful trace of* g.

Definition 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.

Definition 19

A goal clause $g_1, \ldots, g_n$ is true on $\langle I_\perp, I_\infty \rangle$ 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 $t_1, \ldots, t_n$. In such a case, we say that σ *makes* $g_1, \ldots, g_n$ *true* on $\langle I_\perp, I_\infty \rangle$.

The empty goal clause is false on any $\langle I_\perp, I_\infty \rangle$.

Definition 20

A guarded clause,

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

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

(1) If $\{U_{b1}, \ldots, U_{bh}\}$ instantiates a variable which is visible from outside through H, then

$$H : -\{< \{U_{g1}, \ldots, U_{gm}\} | \perp >\} \in I_\perp .$$

(2) Let $Inst_\sigma$ be a set of variables which are instantiated by the substitution σ. If there exists a variable $x \in Inst_{\{U_{g1}, \ldots, U_{gm}\}} \cap$

$Inst_{\{U_{b1},\ldots,U_{bh}\}}$ such that $\{U_{g1},\ldots,U_{gm}\}x$ and $\{U_{b1},\ldots,U_{bh}\}x$ cannot be unified, then

$$H:-\{<\{U_{g1},\ldots,U_{gm}\}|\perp>\}\in I_\perp.$$

(3) If there exists an $\omega-$substitution, σ which does not instantiate variables which are invisible from outside through H and makes $B_1,\ldots,B_n$ true on $\langle I_\perp, I_\infty\rangle$, then there is an I/O history in $I_\perp$ such that,

$$\begin{aligned}H:-\{&<\{U_{g1},\ldots,U_{gm},\}|U_{b1}>,\ldots,<\{U_{g1},\ldots,U_{gm},\}|U_{bh}>\}\cup\\&((GU_1\|\ldots\|GU_n)\bowtie\{U_{g1},\ldots,U_{gm}\})\downarrow Var(H)\end{aligned}$$

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

[Murakami 88] presented the notion of the $\infty-$model of D, that is the set of I/O histories which represent normal computations on D. There exists a unique maximum $\infty-$model for a given D. The maximum model is the *$\omega-$success set of D* and is denoted as M^D_∞.

Definition 21

Let D be a GHC program and M^D_∞ be the $\omega-$success set of D. The $\perp$ interpretation, $I_\perp$ is a *$\perp$model of D* if all clauses in D are true on $\langle I_\perp, M^D_\infty\rangle$.

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

Proposition

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 the proposition, 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 set* of D and is denoted as $M^D_\perp$. The failure set semantics of D is defined with $\langle M^D_\perp, M^D_\infty\rangle$.

For given GHC program D, if a goal clause $g_1,\ldots,g_n$ is true on the failure set semantics of D, then $g_1,\ldots,g_n$ has the potential of finite failure with the program for some input.

4 Fixpoint Semantics

In this section, we will show that the failure set of program D is characterized with the least fixpoint of the function obtained from D and M_{∞}^{D}.

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.

Definition 22

Let D be a GHC program. $Base_D$ is the set defined as follows.

$$\begin{aligned} Base_D = \{H :- \{< \sigma \mid \perp >\} \mid &\text{ for } H :- U_{g1}, \ldots, U_{gm} | U_{b1}, \ldots, U_{bh}, B_1, \ldots, B_n \in D \\ & U_{b1}, \ldots, U_{bh} \text{ instantiates a variable which is visible} \\ & \text{from outside through } H \text{ or there exists a variable} \\ & \quad x \in Inst_{\{U_{g1},\ldots,U_{gm}\}} \cap Inst_{\{U_{b1},\ldots,U_{bh}\}} \\ & \text{and } \{U_{g1}, \ldots, U_{gm}\}x \text{ and } \{U_{b1}, \ldots, U_{bh}\}x \text{ cannot be unified.}\} \end{aligned}$$

Then $\Psi_D : IP \to IP$ is the function defined as follows.

$$\begin{aligned} \Psi_D(S) = Base_D \cup & \\ & \{H :- \{< \{U_{g1}, \ldots, U_{gm},\} | U_{b1} >, \ldots, < \{U_{g1}, \ldots, U_{gm},\} | U_{bh} >\} \cup \\ & \quad ((GU_1 \| \ldots \| GU_n) \ltimes \{U_{g1}, \ldots, U_{gm}\}) \downarrow Var(H) | \\ & \text{for some clause in } D, \\ & H :- U_{g1}, \ldots, U_{gm} | U_{b1}, \ldots, U_{bh}, B_1, \ldots, B_n, \\ & \text{there exists an } \omega-\text{substitution } \sigma \text{ such that} \\ & \sigma \text{ does not instantiate any variables} \\ & \text{which are invisible from outside through } H \\ & \text{and makes } B_1, \ldots, B_n \text{ true on } \langle S, M_{\infty}^{D} \rangle, \text{ and} \\ & GU_i \text{ is the body part of the trace } (\in S \cup M_{\infty}^{D}) \text{ of } \sigma B_i.\} \end{aligned}$$

It is easy to show from the definition of Ψ_D that Ψ_D is continuous from above and $\Psi_D(S) \subset S$ if and only if S is a $\perp$ $-$model of D.

Thus, we derive the following theorem.

Theorem

Let D be a program and $M_{\perp}^{D}$ be the failure set of D. Then,

$$M_{\perp}^{D} = \bigcup\{\Psi_D^n(\phi) | n \geq 0\}.$$

5 Conclusion

This paper presented a new model theoretic semantics for Flat GHC programs based on the failure set. In this paper, we defined that a goal is true on the failure set semantics if there is a substitution which makes the goal to have the potential of finite failure. The purpose of the semantics presented here is the base of discussion for freedom from abnormal execution of a program. Thus a method to discuss whether the ω-substitution which makes the goal true instantiates the output variables or not is important to make the usefulness of the failure set semantics clear. This has been left for the future research as the model theoretic and fixpoint characterization of deadlock.

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, Tech. 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

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

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

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

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

[Ueda 89] K. Ueda, Guarded Horn Clauses, MIT Press, 1988