Vol. 37 No. 8 Transactions of Information Processing Society of Japan Aug. 1996

Regular Paper

An OR-compositional Semantics of GHC for Programs with Perpetual Processes

TORU KATO† and MASAKI MURAKAMI†

OR-compositionality is an important property of semantics of programming language. Many results on the semantics of concurrent logic programming language were reported[1),3),4),7),8),10),11)]. But most of them were not OR-compositional or even if OR-compositional, they required a rigid syntactic restriction. On the other hand an OR-compositional semantics of pure logic programming languages is reported in Ref. 2). That semantics required no syntactic restriction for predicates. It achieved the goal by adding new elements to the semantic domain. Those elements have information for predicates of which definition may be changed by OR-composition operation. We propose an OR-compositional semantics of GHC (Guarded Horn Clauses) for programs with perpetual processes by extending the semantics constructed in Ref. 11) using the idea of Ref. 2).

1. Introduction

Let $\mathcal{O}$ be a semantic mapping. For programs P_α and P_β, we build a new program $P_\alpha * P_\beta$ by applying syntactic operation $*$ to them. If there exists a semantic operation $*_{Den}$ such that $\mathcal{O}(P_\alpha * P_\beta) = \mathcal{O}(P_\alpha) *_{Den} \mathcal{O}(P_\beta)$ then we regard this semantics *compositional* w.r.t. operation $*$. This property indicates that we can construct the semantics of the program $P_\alpha * P_\beta : \mathcal{O}(P_\alpha * P_\beta)$ from the semantics of its components $\mathcal{O}(P_\alpha)$ and $\mathcal{O}(P_\beta)$.

This property allows us to replace $P_\alpha * P_\beta$ with $P'_\alpha * P_\beta$ if $\mathcal{O}(P_\alpha) = \mathcal{O}(P'_\alpha)$ for some program components P_α, P'_α and for a program component P_β. Otherwise it is impossible to replace program components with another components that have same semantics without affecting the semantics of the whole program. Consequently compositionality is important property when we use the semantics as the basis of transformation, composition, or verification of programs.

Our interest is OR-composition operation for concurrent logic programming languages in this paper. This operation is defined as the union of sets of clauses.

Many results on the semantics of concurrent logic programming language were reported[1),3),4),7),8),10),11)]. But most of them were not OR-compositional or even if OR-compositional they required a rigid syntactic restriction. For example the semantics of GHC (Guarded Horn Clauses) was reported in Ref. 11). Though that semantics could express the behavior of perpetual processes, that had no compositionality w.r.t. the OR-composition operation (we will abbreviate it to OR-compositionality in this paper). The semantics of CC (Concurrent Constraint) which is OR-compositional was reported in Ref. 6). But it required a rigid syntactic restriction such that the OR-composition of programs is defined only if the common predicates among programs must have the same structure.

On the other hand an OR-compositional semantics of pure logic programming languages without mechanism for parallel processing is reported in Ref. 2). That semantics required no syntactic restriction for predicates. It achieved the goal by adding new elements to the semantic domain. Those elements have informations for predicates of which definition may be changed by OR-composition operation.

We propose an OR-compositional semantics of GHC for programs with perpetual processes by extending the semantics constructed in Ref. 11) using the idea of Ref. 2). GHC have a synchronizing mechanism and it prevent us to apply just the same means of Ref. 2) to the semantics of GHC. So we apply an extended idea of Ref. 2) to our semantics.

We extend the idea of Ω open program defined in Ref. 2) for GHC.

Definition 1.1 (Term) Let *Var* be a set of variables, $Fun = \{a, b, nil, cons\}$ a set of function symbols where a, b and nil are 0-ary symbols and $cons$ a 2-ary symbol. We define the

† Faculty of Engineering, Okayama University

set *Terms* and *term* as follows:

i. if $\tau \in Var$ or $\tau \in Fun$, then $\tau \in Terms$

ii. if $\tau_1, \tau_2 \in Terms$;, then $cons(\tau_1, \tau 2) \in Terms$

We call a element of the *Terms* a *term*.

Definition 1.2 (Unification goal) '$X = \tau$' is a unification goal where $X \in Var$, τ is a term. We denote a unification goal $X = X$ as *true* especially.

A set σ of unification goals defines a substitution which is obtained when all unifications succeed. For example we can regard the set $\sigma = \{X = [A|Y], A = a\}$ as a substitution that maps A to a and X to $[a|Y]$. We will identify a set of unification goals with the substitution defined by it. In other words the equation such as $\sigma_1 = \sigma_2$ indicates that σ_1 and σ_2 define the same substitution[11].

Definition 1.3 (Function Pred, Var) Let P be a Flat GHC program[11], a clause of Flat GHC program or an atom. **Pred**(P) is the set of all predicate symbols appear in P and **Var**(P) is the set of all function symbols appear in P.

Definition 1.4 (Ω open program) Let P be a program and Ω is a set of predicate symbols. P is an Ω open program if there exists the following clauses in P:

$$h{:-}\sigma|U_1, \ldots, U_l, b_1, \ldots, b_n.$$

where each U_i is unification goal, each b_j is atom other than an unification goal, $\exists m(1 \leq m \leq n).\mathbf{Pred}(b_m) \subseteq \Omega$.

Definition 1.5 (OR-composition) Let P_α and P_β be Ω open programs. If $\mathbf{Pred}(P_\alpha) \cap \mathbf{Pred}(P_\beta) \subseteq \Omega$ then the OR-composition of P_α and P_β is defined as the program $P_\alpha \cup P_\beta$. Otherwise the OR-composition of P_α and P_β is not defined.

Example 1.6 *We present examples of Ω open program.*

P_A={$p(X)$:$-X = [A|X_1]|X_1 = [a|X_2], q(X_1)$.
$q(X)$:$-X = [A|X_1]|X_1 = []$.}.
P_B={$q(X)$:$-X = [A|X_1]|X_1 = [b|X_2], p(X_1)$.
$p(X)$:$-X = [A|X_1]|X_1 = []$.}.

Let $p, q \in \Omega$, then P_A and P_B are Ω open programs because they satisfy the condition of Definition 1.4. *The program $P_A \cup P_B$ can produce a perpetual process. For example there exists a process $p(X)$ which outputs the infinite list such as $a, b, a, b, \ldots$. This behavior is represented by the following I/O history*[11]:

$$p(X) :- \{\langle X = [A|X_1]|X_1 = [a|X_2]\rangle, \langle X = [A|X_1]|X_2 = [b|X_3]\rangle, \ldots\}. \quad (1)$$

This paper defines the new semantic mapping and new OR-composition operation on the semantic domain which satisfy OR-compositionality even if a program produces perpetual processes as Example 1.6.

2. Semantic Domain

This section introduces the semantic domain and the semantic mapping which can map a program with perpetual processes to the element of the domain.

2.1 I/O History

Definition 2.1 (Relation $\models$[11]) Let σ be a set of unification goals and U a unification goal. If $\sigma \cup \{U\}$ defines a substitution then U is consistent with σ. Furthermore if $\sigma \cup \{U\}$ defines the same substitution to σ then we denote $\sigma \models U$.

Definition 2.2 (Simple substitution) A substitution $X = \tau$ is simple if τ is a variable or a 0-ary symbol or has the form of $\mathrm{cons}(X, Y)$ where X and Y are different variables.

Definition 2.3 (ω substitution[11]) Let σ be a set of simple substitutions. If σ is a substitution or equal to $\bigcup_{k=0}^{\infty} \theta_k$ such that $\theta_0 = \emptyset$ and $\theta_{k+1} = \theta_k \cup \{X = \tau\}$ for some $X = \tau$ which is consistent with θ_k and $\theta_k \not\models (X = \tau)$ then σ is an ω-substitution.

Definition 2.4 (Guarded unification) Tuple $\langle \sigma | U \rangle$ and tuple $\langle \sigma, \tilde{b} | true \rangle$ are guarded unifications where σ is an ω-substitution, U is a unification goal, and $\tilde{b}$ is a tuple of atoms other than a unification goal. Especially we call $\langle \sigma, \tilde{b} | true \rangle$ extended guarded unification if necessary.

Intuitively $\langle \sigma | U \rangle$ represents the event such that the unification goal U is executed after the arguments of a goal are instantiated by σ. This event corresponds to the commit operation of GHC program[12]. $\langle \sigma, \tilde{b} | true \rangle$ indicates the event that the goal clause $\tilde{b}$ is called after the arguments of the goal are instantiated by σ.

Definition 2.5 For a guarded unification $\langle \sigma | U \rangle$ or $\langle \sigma, \tilde{b} | true \rangle$, we call σ as input unification or guard part of guarded unification. And we call U or *true* as output unification. For a guarded unification gu, **In**(gu) represents a input unification of gu and **Out**(gu) represents a output unification of gu.

Definition 2.6 (Function $|\cdot|$[11]) Let gu be a guarded unification. $|gu|$ is a substitution as follows:

$$|gu| = \mathbf{In}(gu) \cup \{\mathbf{Out}(gu)\}.$$

Let GU be a set of guarded unifications. $|GU|$ is a substitution as follows:

$$|GU| = \bigcup_{gu \in GU} |gu|.$$

Definition 2.7 (binary relation $\prec$[11]) Let gu_1 and gu_2 be guarded unifications. Following relation holds if and only if there exists a substitution θ_1 such that $\mathbf{In}(gu_1)\theta_1 = \mathbf{In}(gu_2)$ and there is no substitution θ_2 such that $\mathbf{In}(gu_1) = \mathbf{In}(gu_2)\theta_2$:

$$gu_1 \prec gu_2.$$

Where $\sigma\theta$ is a composition of substitutions σ and θ[9].

Intuitively $\langle \sigma_1 | U_1 \rangle \prec \langle \sigma_2 | U_2 \rangle$ indicates that U_2 requires more instantiation than U_1 to be executable. In other words, U_1 has already been executable when U_2 turns executable.

Definition 2.8 (Closed from below[11]) Let GU be a set of guarded unifications and Gu be a finite subset of GU. Gu is closed from below iff for any $gu, gu' \in GU$ such that $gu' \prec gu$,

$$gu \in Gu \rightarrow gu' \in Gu.$$

Definition 2.9 (Frontier) Let GU be a set of guarded unifications and Gu be a finite subset of GU that is closed from below. A guarded unification $gu \in GU$ is a frontier of Gu if for any $gu' \in Gu$,

$$gu \not\prec gu' \text{ and } gu' \neq gu.$$

And gu is a minimal frontier of Gu if there is no $gu'' \in GU$ such that $gu'' \prec gu$.

Definition 2.10 (Guarded stream[11]) A set of guarded unifications GU is a guarded stream if the followings are true:

For any finite subset Gu of GU that is closed from below and for any minimal frontier gu of Gu,

i. $\mathbf{Out}(gu)$ is consistent with $|Gu| \cup \mathbf{In}(gu)$.

ii. For any $U' \in \mathbf{In}(gu)$, U' is consistent with $|Gu|$ and there is no substitution θ such that $|Gu| = \theta\mathbf{In}(gu)$.

Intuitively i ii above define the condition to represent a set of events occurring in an execution of a GHC program. Condition i indicates that there is no computation which waits for inputs contradicting with bindings given from outside of the processes or with bindings already computed by the process. Condition ii indicates that new output bindings are not included by bindings nor are contradicted by bindings given from outside of the process or bindings already calculated by the process.

Definition 2.11 (I/O-history) Let $h \equiv p(X_1, \ldots X_n)$ and GU be a guarded stream. $h :- GU$ is an I/O-history.

Definition 2.12 (trace[11]) Let $t : h{:-}GU$ be an I/O-history and g be a goal clause. t is a trace of g if for some substitution θ, following conditions hold:

- $h\theta = g$
- $\forall gu \in GU.\mathbf{In}(gu) \subseteq \theta$.
- $\forall \langle \sigma | X = \tau \rangle \in GU.(\theta$ does not instantiate X or $X\theta = \tau\theta)$.

2.2 Semantic Mapping

This paper adopts the greatest fixpoint semantics so that we can express I/O-histories with infinite length which are associated with perpetual processes. The semantic mapping presented here is defined as extraction of I/O-histories which express calculations of a program from the set of all I/O-histories including ones with infinite length.

The synchronized merge operation '$\|$'[11] enables us to get one of the guarded streams for a clause from guarded streams for subgoals of the clause. We can regard the operation as an extended concept of the combination of substitutions.

In this paper we use the extended notion of guarded unification (Definition 2.4), so the definition of synchronized merge must be extended. Before defining the extended synchronized merge operation, we present some definition needed for synchronized merge definition.

Definition 2.13 (Predicate ρ) Let $GU_1, \ldots, GU_n$ be guarded streams and σ be a finite set of unification goals. We define a predicate ρ as follows:

$$\rho(\sigma, GU_1, \ldots, GU_n) \stackrel{def}{\equiv}$$
$$\forall U' \in \sigma, \forall Gu \subset \bigcup_{j=1}^{n} GU_j.|Gu| \models U' \Rightarrow$$
$$(|Gu| \setminus \{X = \tau | \exists \langle \sigma' | X = \tau \rangle \in Gu\}) \models U'.$$

Definition 2.14 (Function G) Let $GU_1, \ldots, GU_n$ be guarded streams and S_1 be a set of guarded unifications. We define a set of guarded unifications $\mathbf{G}(GU_1, \ldots, GU_n, S_1)$ as follows:

$$\mathbf{G}(GU_1, \ldots, GU_n, S_1) =$$
$$\{gu | \ \exists i, gu \in GU_i.$$
$$(\forall U' \in \mathbf{In}(gu), \exists S_2 \subset S_1.|S_2| \models U'$$
$$\vee \rho(\mathbf{In}(gu), GU_1, \ldots, GU_n))\}.$$

Definition 2.15 (Function Σ) Let gu be a guarded unification and S_1 be a set of guarded unifications. We define a substitution $\mathbf{\Sigma}(gu, S_1)$ as follows:

$\mathbf{\Sigma}(gu, S_1) =$
$(\mathbf{In}(gu) \setminus \{U' \mid |S_1| \models U'\}) \cup$
$\{U' | \exists S_2 \subset S_1, \exists \langle \sigma' | U'' \rangle \in S_2,\ U' \in \sigma' \wedge$
$(\exists U''' \in \mathbf{In}(gu).(|S_2| \models U''' \wedge$
$\forall \theta \subseteq |S_2|.|S_2| \neq \theta \Rightarrow \theta \not\models U'''))\}.$

Definition 2.16 (Synchronized merge) Let $GU_1, \ldots, GU_n$ be sets of guarded unifications and Gu_k $(0 \leq k)$ be sets of guarded unifications as follows:

$Gu_0 =$
$\{gu | \exists i, gu \in GU_i.\rho(\mathbf{In}(gu), GU_1, \ldots, GU_n)\}.$
$Gu_{k+1} = Gu_k \cup$
$\{\langle \Sigma | \mathbf{Out}(gu) \rangle |\ gu \in \mathbf{G}(GU_1, \ldots, GU_n, Gu_k),$
$\Sigma = \begin{cases} \mathbf{\Sigma}(gu, Gu_k) & \text{if } gu = \langle \sigma | U \rangle, \\ \mathbf{\Sigma}(gu, Gu_k), \tilde{b} & \text{if } gu = \langle \sigma, \tilde{b} | true \rangle \end{cases}$
$\}.$

Let $GU = \bigcup_{k>0} Gu_k$. If GU is guarded stream and $\{U | \exists \langle \Sigma | U \rangle \in GU\} = \{U | \exists i, \exists \langle \Sigma | U \rangle \in GU_i\}$ then GU is a *synchronized merge* of $GU_1, \ldots, GU_n$ and we denote it as
$GU_1 \| \ldots \| GU_n$.

Intuitively the roles of $\rho, \mathbf{G}$ and $\mathbf{\Sigma}$ in Definition 2.16 are explained as follows. A synchronized merge $GU_1 \| \ldots \| GU_n$ is constructed by generating the sets of guarded unifications $Gu_1, Gu_2, ..., Gu_k, ...$ successively. For all guarded unification in Gu_k there exists a guarded unification $\langle I | O \rangle$ in $GU_1, ...GU_n$ and it requires at most k times of communications in $GU_1, ...GU_n$ to solve I. Gu_0 is a set of guarded unifications whose guard parts are satisfied by 0 times of communication. In other words those guard parts are satisfied by the inputs from outside of $GU_1, ...GU_n$. $\rho(\sigma, GU_1, .., GU_n)$ defines the condition that it requires no output from $GU_1, \ldots, GU_n$ to solve σ. This means σ is solved by the input from outside of $GU_1, ...GU_n$.

Every guarded unification in Gu_{k+1} is obtained from a guarded unification $\langle I | O \rangle$ in $GU_1, ...GU_n$ such that I is solved by outputs of $GU_1, \ldots, GU_n$ or inputs from the outside of $GU_1, \ldots, GU_n$. $\mathbf{G}(GU_1, ...GU_n, Gu_k) (\subseteq GU_1 \cup \ldots \cup GU_n)$ is a set of such guarded unifications as $\langle I | O \rangle$ above.

Let $gu \in \mathbf{G}(GU_1, \ldots, GU_n, Gu_k)$. $\mathbf{\Sigma}(gu, Gu_k)$ define the operation replacing unifications in guard part of gu satisfied by the output $X = \tau$ from $GU_1, ...GU_n$ with input unifications from outside required to execute $X = \tau$.

We also extend $\downarrow$ operation[11] and $\bowtie$ operation[11] as follows.

Definition 2.17 (Function $\downarrow$) Let GU be a guarded stream and $V \subset Var$. The restriction of GU by V : $GU \downarrow V$ is the set defined as follows:
$GU \downarrow V =$
$\{\langle \sigma | X = \tau \rangle | \exists k, \langle \sigma | X = \tau \rangle \in GU, X \in V_k\}$
$\cup \{\langle \sigma', \tilde{b} | true \rangle | \langle \sigma', \tilde{b} | true \rangle \in GU\},$
where
$V_0 = V,$
$V_{i+1} = V_i \cup$
$\{X | \exists gu \in GU, \exists (Y = \tau) \in |gu|, \exists Y \in V_i.$
X appears in $\tau, \forall gu' \in GU$, if $gu' \prec gu$
X then does not occur in $gu'\}.$

Intuitively the $\downarrow$operator removes guarded unifications whose left hand side variable of output unification is used as inner shared variable. Consequently I/O-history $h{:-}GU \downarrow$ which represents a process contains only guarded unifications whoes output action can be observed from the environment of the process.

Definition 2.18 (Function $\bowtie$) Let GU be a guarded stream, σ_1, σ_2 be substitutions, $X \in Var$ and τ is a term. The set $GU \bowtie (\sigma_1, \sigma_2)$ is defined as follows:

$GU \bowtie (\sigma_1, \sigma_2) =$
$\{\langle \Sigma | U_b \rangle | \exists \langle \sigma' | U_b' \rangle \in GU.\ \Sigma = \sigma' \cup \sigma_1 \setminus \sigma_2,$
$U_b = \begin{cases} U_b' & \text{if } \sigma_1 \cup \sigma_2 \not\models U_b' \\ true & \text{otherwise} \end{cases}$
$\} \cup \{\langle \Sigma, \tilde{b} | true \rangle |\ \exists \langle \sigma', \tilde{b} | true \rangle \in GU,$
$\Sigma = \sigma' \cup \sigma_1 \setminus \sigma_2\}.$

Intuitively the $\bowtie$ operator removes unifications in σ_2 from input unifications of all guarded unification in GU and adds unifications in σ_1 to input unifications of all guarded unification in GU.

Definition 2.19 (I/O-hist) A renaming mapping over the domain of I/O-histories defines an equivalence relation. I/O-hist is the quotient set defined by this equivalence relation. From now on an element of I/O-hist is I/O-history afresh. IP is the power set of I/O-hist .

In this paper when all elements of a tuple $\tilde{X}$ are contained in a set Y, we express it as $\tilde{X} \subset Y$. When all elements of a tuple $\tilde{X}$ are contained in a tuple $\tilde{Z}$ and $\tilde{X}$ keeps the order of the elements in $\tilde{Z}$, we express it as $\tilde{X} \subset \tilde{Z}$ and we call $\tilde{X}$ subtuple of $\tilde{Z}$. When x is an element of tuple $\tilde{X}$, we express it as $x \in \tilde{X}$. From now on we describe a tuple by enclosing with "[ ]" so that we can discriminate a tuple from a set enclosed by " {}".

Definition 2.20 (conGs, Gs) Let C be a clause as follows:

$C = h{:-}\sigma|U_1,\ldots,U_n,b_1,\ldots,b_k,\ldots,b_l$

where each U_i is unification goal, each b_j is atom other than an unification goal. For some $j \subseteq \{k,\ldots,l\}$, let $\widetilde{io} = [b_i'{:-}GU_i|\exists i \in \{1,\ldots,k-1\}\cup j]$ be a tuple of I/O-history. For $C, \widetilde{io}$ *and* $j \subseteq \{k,\ldots,l\}$, we define the predicate $\mathbf{conGs}(C,j,\widetilde{io})$ a set of guarded unification $\widehat{\mathbf{Gs}}(C,j,\widetilde{io})$ and guarded stream $\mathbf{Gs}(C,j,\widetilde{io})$ as follows:

$\mathbf{conGs}(C,j,\widetilde{io}) \stackrel{\text{def}}{\equiv}$
$\exists\theta,\ \forall i(\in\{1,\ldots,k-1\}\cup j).$
$b_i'{:-}GU_i(\in\widetilde{io})$ is a trace of $b_i\theta$.

$\widehat{\mathbf{Gs}}(C,j,\widetilde{io}) =$
$\{\langle\sigma|U_1\rangle,\ldots,\langle\sigma|U_n\rangle\}\cup$
$((GU_1\|\ldots\|GU_{k-1}\|(\|_{t\in j}GU_t))\bowtie$
$(\sigma,\{U_1,\ldots,U_n\}))\downarrow\mathbf{Var}(h)\cup\{\langle\sigma,\widetilde{b}|true\rangle\}.$
where $\widetilde{b} = [b_s|\exists s\in\{k,\ldots,l\}\setminus j]$.

If $\widehat{\mathbf{Gs}}(C,j,\widetilde{io})$ is a guarded stream then $\mathbf{Gs}(C,j,\widetilde{io}) = \widehat{\mathbf{Gs}}(C,j,\widetilde{io})$, else $\mathbf{Gs}(C,j,\widetilde{io})$ is undefined.

Definition 2.21 (Function T_p) Let P be a GHC Ω open program, $C(\in P)$ a clause as $h{:-}\sigma|U_1,\ldots,U_n,b_1,\ldots,b_k,\ldots,b_l$, where σ is a set of unification-goals, each U_i is unification-goal, each b_i is atom other than a unification goal and $\mathbf{Pred}(b_k),\ldots,\mathbf{Pred}(b_l)\subseteq\Omega$.
$T_P : IP \to IP$ is the function defined as follows:

$T_P(S) = S\cap$
$\{h{:-}\mathbf{Gs}(C,j,\widetilde{io})|\ C\in P, j\subseteq\{k,\ldots,l\},$
$\widetilde{io}\subseteq S, \mathbf{conGs}(C,j,\widetilde{io})\}.$

For an I/O-history $t = h{:-}\mathbf{Gs}(C,j,\widetilde{io})$, we call a clause C the associating clause of t.

Definition 2.22 (Function $\mathcal{O}_\Omega$) Let P be a GHC Ω open program, $T_P\downarrow 0 =$ I/O-hist and $T_P\downarrow(n+1) = T_P(T_P\downarrow n)$. The semantic mapping $\mathcal{O}_\Omega$ is defined as follows:

$$\mathcal{O}_\Omega(P) = \bigcap_{n\geq 0} T_P\downarrow n\,.$$

Example 2.23 *Let P_A and P_B be Ω open programs defined in* Example 1.6 *and let* $P = P_A\cup P_B$.
$\mathcal{O}_\Omega(P) = \bigcap_{n\geq 0} T_P\downarrow n$ *where*
$T_P\downarrow 0 =$
$\{p(X){:-}\{\langle X=[A|X_1]|X_1=[a|X_2]\rangle,\ldots\}.,$
$\vdots$
$q(X){:-}\{\langle X=[A|X_1]|X_1=[b|X_2]\rangle,\ldots\}.,\ldots\}$
$T_P\downarrow 1 =$
$\{\ p(X){:-}\{\ \langle X=[A|X_1]|X_1=[a|X_2]\rangle,$
$\langle X=[A|X_1]|X_2=[b|X_3]\rangle,\ldots\}.,$
$\vdots$
$q(X){:-}\{\ \langle X=[A|X_1]|X_1=[b|X_2]\rangle,$
$\langle X=[A|X_1]|X_2=[a|X_3]\rangle,\ldots\}.,$
$\vdots\}\ldots$
$T_P\downarrow n =$
$\{\ p(X){:-}\{\ \langle X=[A|X_1]|X_1=[a|X_2]\rangle,$
$\langle X=[A|X_1]|X_2=[b|X_3]\rangle,\ldots,$
$\langle X=[A|X_1]|X_{2i}=[b|X_{2i+1}]\rangle,$
$\langle X=[A|X_1]|X_{2i+1}=[a|X_{2i+2}]\rangle,$
$\ldots\}.\ \ldots\}\ldots$

Repeating the same operation, we can show (1) *of* Example1.6 *is an element of* $\mathcal{O}_\Omega(P_A\cup P_B)$ *inductively.*

3. OR-composition Operation

The OR-composition operation of $\mathcal{O}_\Omega(P_A)$ and $\mathcal{O}_\Omega(P_B)$ is defined as the extraction of I/O-histories expressing the computations on the program $P_A\cup P_B$ from the set of all I/O-histories. This operation can construct I/O-histories with infinite length expressing the infinite computations on $P_A\cup P_B$.

We extend the domain of function **Pred** (Definition 1.3) to contain semantics of program.

Definition 3.1 $\mathbf{Pred}(\mathcal{O}(P))$ is the set of all predicate symbols appear in $\mathcal{O}(P)$.

Definition 3.2 ($\mathbf{Gs_{OR}}$) Let t be an I/O-history as follows:

$t = h{:-}\{\ \langle\sigma_1|U_1\rangle,\ldots,\langle\sigma_n|U_n\rangle,\ldots$
$\langle\sigma_{n+1},\widetilde{b_1}|true\rangle,\ldots,$
$\langle\sigma_{n+m},\widetilde{b_m}|true\rangle,\ldots\}$
where $\widetilde{b_i} = [b_i^1,\ldots,b_i^{p_i}]$,
$\{\mathbf{Pred}(b_i^1),\ldots,\mathbf{Pred}(b_i^{p_i})\}\subseteq\Omega.$

And let $\mathcal{J}$ be a set of J (tuple of suffixes of goals) as follows: $\mathcal{J} = \{J|1\leq i<\infty,\exists j_i\in 2^{\{1,\ldots,p_i\}},\ J=[j_1,\ldots,j_m,\ldots]\}$. We define the set of guarded unifications $\widehat{\mathbf{Gs_{OR}}}(t,J,\widetilde{io})$ and the guarded stream $\mathbf{Gs_{OR}}(t,J,\widetilde{io})$ for some $J\in\mathcal{J}$, t and a tuple of I/O-history $\widetilde{io} = [b_i'^p{:-}G_i'^p|1\leq i<\infty,\exists j_i\in J,\exists p\in j_i]$ as follows:

$\widehat{\mathbf{Gs_{OR}}}(t,J,\widetilde{io}) =$
$\{\langle\sigma_1|U_1\rangle,\ldots,\langle\sigma_n|U_n\rangle\}\cup(\|_{i\geq 1}(\|_{p\in j_i}G_i^p$
$\bowtie(\sigma_{n+i},\{U_1,\ldots,U_n\})\downarrow\mathbf{Var}(h))$
$\cup(\cup_{i\geq 1}\{\langle\sigma_{n+i},\widetilde{b_{t_i}}|true\rangle\}).$

where $\widetilde{b_{t_i}}$ is a tuple as

$$\widetilde{b_{t_i}} = [b_i^v | v \in \{1, \ldots, p_i\} \setminus j_i].$$

If $\widehat{\mathbf{Gs_{or}}}(t, J, \widetilde{io})$ is a guarded stream then $\mathbf{Gs_{or}}(t, J, \widetilde{io}) = \widehat{\mathbf{Gs_{or}}}(t, J, \widetilde{io})$, else $\mathbf{Gs_{or}}(t, J, \widetilde{io})$ is undefined.

Definition 3.3 (Operation $\bigcup_\Omega$) Let P_A and P_B be Ω open programs. If $\mathbf{Pred}(\mathcal{O}_\Omega(P_A)) \cap \mathbf{Pred}(\mathcal{O}_\Omega(P_B)) \not\subseteq \Omega$ then the OR-composition on the semantic domain is undefined likewise the case on the syntactic domain. Otherwise the OR-composition operation $\bigcup_\Omega$ is defined as follows:

$\mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B) = \bigcap_{k \geq 0} H_k$

where

$H_0 =$ I/O-hist,

$H_{k+1} = H_k \cap$

$\{h{:-}\mathbf{Gs_{or}}(t, J, \widetilde{io}) |$

$t \in \mathcal{O}_\Omega(P_A) \cup \mathcal{O}_\Omega(P_B), J \in \mathcal{J},$

$\widetilde{io} = \{b_i^{'p}{:-}G_i^{'p} | 1 \leq i < \infty, \exists\theta, \exists j_i \in J,$

$p \in j_i, b_i^{'p}{:-}G_i^{'p} \in H_k.$

$b_i^{'p}{:-}G_i^{'p}$ is a trace of $b_i^p\theta\}\}$.

For an I/O-history $t = h{:-}\mathbf{Gs_{or}}(t', J, \widetilde{io})$, we call an I/O-history t' the associating I/O-history of t and we say t is constructed from t'.

Example 3.4 *Let P_A and P_B be GHC Ω open programs defined in* Example 1.6.

$\mathcal{O}_\Omega(P_A) =$

$\{ q(X){:-}\{ \langle X = [A|X_1] | X_1 = [] \rangle\}.$

$p(X){:-}\{ \langle X = [A|X_1] | X_1 = [a|X_2] \rangle,$

$\langle X = [A|X_1] | X_2 = [] \rangle\}.$

$p(X){:-}\{ \langle X = [A|X_1] | X_1 = [a|X_2] \rangle,$

$\langle X = [A|X_1], q(X_1) | true \rangle\}.\}.$

$\mathcal{O}_\Omega(P_B) =$

$\{ p(X){:-}\{ \langle X = [A|X_1] | X_1 = [] \rangle \}.$

$q(X){:-}\{ \langle X = [A|X_1] | X_1 = [b|X_2] \rangle,$

$\langle X = [A|X_1] | X_2 = [] \rangle\}.$

$q(X){:-}\{ \langle X = [A|X_1] | X_1 = [b|X_2] \rangle,$

$\langle X = [A|X_1], p(X_1) | true \rangle\}.\}.$

$\mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B) = \bigcap_{k \geq 0} H_k$ where

$H_0 =$

$\{ p(X){:-}\{\langle X = [A|X_1] | X_1 = [a|X_2] \rangle, \ldots\}.,$

$\vdots$

$q(X){:-}\{\langle X = [A|X_1] | X_1 = [b|X_2] \rangle, \ldots\}.,$

$\vdots\} \ldots$

$H_1 =$

$\{ p(X){:-}\{ \langle X = [A|X_1] | X_1 = [a|X_2] \rangle,$

$\langle X = [A|X_1] | X_2 = [b|X_3] \rangle, \ldots\}.,$

$\vdots$

$q(X){:-}\{ \langle X = [A|X_1] | X_1 = [b|X_2] \rangle,$

$\langle X = [A|X_1] | X_2 = [a|X_3] \rangle, \ldots\}.,$

$\vdots\} \ldots$

$H_n =$

$\{ p(X){:-}\{ \langle X = [A|X_1] | X_1 = [a|X_2] \rangle,$

$\langle X = [A|X_1] | X_2 = [b|X_3] \rangle, \ldots,$

$\langle X = [A|X_1] | X_{2i} = [b|X_{2i+1}] \rangle,$

$\langle X = [A|X_1] | X_{2i+1} = [a|X_{2i+2}] \rangle,$

$\ldots\}.\ldots\} \ldots$

Repeating the same operation, we can show (1) *of* Example 1.6 *is an element of $\mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B)$ inductively.*

By the way the I/O-histories constructed here are identified with the I/O-histories constructed in Example 2.23. *This indicates that* Definition 3.3 *is in accordance with* Definition 2.22.

4. Justification of OR-composition Operation

In this section we prove that our semantics satisfy compositionality. More strictly, for any GHC Ω open programs P_A and P_B, we prove

$$\mathcal{O}_\Omega(P_A \cup P_B) = \mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B).$$

Lemma 4.1 For any GHC Ω open programs P_A and P_B,

$$\mathcal{O}_\Omega(P_A \cup P_B) \supseteq \mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B).$$

Proof: Let $P = P_A \cup P_B$, T_P be a function of Definition 2.21. We show I/O-histories that are not contained in $\mathcal{O}_\Omega(P_A \cup P_B)$ are also not contained in $\mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B)$. For y (≥ 0), we define the set D_y be $D_y = T_P{\downarrow}y \setminus T_P{\downarrow}(y+1)$. By Definition 2.21, for any y and for any I/O-history $\bar{t} \in D_y$ if we write $\bar{t}$ as $h{:-}\mathbf{Gs}(C, j, \widetilde{io})$ then the following condition holds:

$$\widetilde{io} \subseteq T_P{\downarrow}y \wedge \mathbf{conGs}(C, j, \widetilde{io}) \rightarrow C \notin P. \quad (2)$$

By Definition 3.3, every I/O-history $t = h{:-}\mathbf{Gs_{or}}(t', J, \widetilde{io'})$ in $\mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B)$ has the associating I/O-history t' in $\mathcal{O}_\Omega(P_A) \cup \mathcal{O}_\Omega(P_B)$. It is obvious from Definition 2.21 and Definition 2.22 that $\mathcal{O}_\Omega(P_A) \cup \mathcal{O}_\Omega(P_B) \subseteq \mathcal{O}_\Omega(P_A \cup P_B)$. By Definition 2.21, all I/O-histories in $\mathcal{O}_\Omega(P_A \cup P_B)$ must have the associating clause in P, so do t'. Hence there exists $C' \in P$ such that

$$t' = h{:-}\mathbf{Gs}(C', j_{t'}, \widetilde{io_{t'}}), \quad (3)$$

where $\mathbf{conGs}(C', j_{t'}, \widetilde{io_{t'}})$ holds.

t' has also following structure:

$$t' = h{:-}\{\ \langle\sigma_1|u_1\rangle, \ldots, \langle\sigma_n|u_n\rangle, \ldots \langle\sigma_{n+1}, \widetilde{b_1}|true\rangle, \ldots, \langle\sigma_{n+m}, \widetilde{b_m}|true\rangle, \ldots\} \quad (4)$$

where $\widetilde{b_i} = [b_i^1, \ldots, b_i^{p_i}]$, $\{\mathbf{Pred}(b_i^1), \ldots, \mathbf{Pred}(b_i^{p_i})\} \subset \Omega$.

Let C' be $h{:-}\sigma|U_1, \ldots, U_n, b_1, \ldots, b_k, \ldots, b_l$, and $\mathbf{Pred}(b_k)), \ldots, \mathbf{Pred}(b_l)) \subseteq \Omega$. If j'_t of (3) is not equal to $\{k, \ldots, l\}$ then for one of extended guarded unifications $\langle\sigma_{n+s}, \widetilde{b_s}|true\rangle \in \{\langle\sigma_{n+1}, \widetilde{b_1}|true\rangle, \ldots, \langle\sigma_{n+m}, \widetilde{b_m}|true\rangle\}$, $\widetilde{b_s}$ is a subtuple of $[b_k, \ldots, b_l]$, and the other extended guarded unifications are elements of bodes of I/O-history in $\widetilde{io_{t'}}$ of (3). t is constructed by merging body of some I/O-histories in $\widetilde{b_v}$ for all $v \in \{1, \ldots, m\}$. So t can be written as follows:

$$t = h{:-}\mathbf{Gs}(C', j_{t''}, \widetilde{io'_{t'}}).$$

Where $j' \subseteq j''$, $\forall io \in \widetilde{io'_{t'}}, \exists io' \in \widetilde{io_{t'}}$, the relation of io and io' is same as t and t' and $\mathbf{conGs}(C', j_{t''}, \widetilde{io'_{t'}})$ holds. Else if j'_t of (3) is equal to $\{k, \ldots, l\}$ then all extended guarded unifications of (4) are elements of bodes of I/O-history in $\widetilde{io_{t'}}$ of (3), and t can be written as follows:

$$t = h{:-}\mathbf{Gs}(C', j_{t'}, \widetilde{io'_{t'}}).$$

In any case, t constructed from t' has the associating clause $C' \in P$. Every I/O-history $\bar{t}$ satisfying the condition (2) is constructed from a clause $C \notin P$.

Consequently I/O-histories $\bar{t}$ is never in $\mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B)$. Thus we proved $D_y \not\subseteq \mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B)$ for any y. Consequently

$$\mathcal{O}_\Omega(P_A \cup P_B) \supseteq \mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B).$$

Lemma 4.2 For any GHC Ω open programs P_A and P_B,

$$\mathcal{O}_\Omega(P_A \cup P_B) \subseteq \mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B).$$

Proof: Let H_y $(y \geq 0)$ be the set defined in Definition 3.3. We show I/O-histories not contained in $\mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B)$ are also not contained in $\mathcal{O}_\Omega(P_A \cup P_B)$.

For any y (≥ 0), we define the set $\mathcal{D}_y$ be $\mathcal{D}_y = H_y \setminus H_{y+1}$ and the set T be

$$T = \{h{:-}G| \exists h{:-}G' \in \mathcal{O}_\Omega(P_A) \cup \mathcal{O}_\Omega(P_B), \exists h{:-}G \in \text{I/O-hist}, G \supseteq G' \setminus \{gu | gu \in G', gu \text{ is an extended guarded unification}\}\}.$$

Intuitively T is a set of all I/O-histories which has the associating clause in $P_A \cup P_B$. By Definition 3.3, for any y and for any I/O-history $\bar{t} \in \mathcal{D}_y$, if we write $\bar{t}$ as $= h{:-}\mathbf{Gs_{or}}(t, J, \widetilde{io})$ then the following condition holds:

$$\widetilde{io} \subseteq H_y \wedge \widetilde{b_i} = [b_i^1, \ldots, b_i^{p_i}] \wedge \{\mathbf{Pred}(b_i^1), \ldots, \mathbf{Pred}(b_i^{p_i})\} \subseteq \Omega \wedge \mathcal{J} = \{J' | 1 \leq i < \infty, j_i \in 2^{\{1,\ldots,p_i\}}, J' = [j_1, \ldots j_m, \ldots]\} \wedge \widetilde{io} = [b_i'^p{:-}G_i'^p| 1 \leq i < \infty, \exists\theta, \exists J \in \mathcal{J}, \exists j_i \in J, p \in j_i, b_i'^p{:-}G_i'^p \text{ is atrace of } b_i^p\theta]$$

$\rightarrow$

$$t = h{:-}\{\ \langle\sigma_1|U_1\rangle, \ldots, \langle\sigma_n|U_n\rangle, \ldots, \langle\sigma_{n+1}, \widetilde{b_1}|true\rangle, \ldots, \langle\sigma_{n+m}, \widetilde{b_m}|true\rangle, \ldots\} \notin T. \quad (5)$$

The reason why we use T instead of $\mathcal{O}_\Omega(P_A) \cup \mathcal{O}_\Omega(P_B)$ is that all I/O-histories constructed from the element of T can be constructed from the element of $\mathcal{O}_\Omega(P_A) \cup \mathcal{O}_\Omega(P_B)$, so I/O-histories constructed from the element of T are not contained in $\mathcal{D}_y$. By Definition 2.21, every I/O-history $t = h{:-}\mathbf{Gs}(C, j, \widetilde{io'}) \in \mathcal{O}_\Omega(P_A \cup P_B)$ is constructed from a certain clause in $P_A \cup P_B$. As $\mathcal{O}_\Omega(P_A) \cup \mathcal{O}_\Omega(P_B) \subseteq \mathcal{O}_\Omega(P_A \cup P_B)$, every t' in $\mathcal{O}_\Omega(P_A) \cup \mathcal{O}_\Omega(P_B)$ has the associating clause C in $P_A \cup P_B$, so do for all I/O-history t'' in T. Namely, for all $t'' \in T$ there exists $C \in P$ such that

$$t'' = h{:-}\mathbf{Gs}(C, j_{t''}, \widetilde{io_{t''}}),$$

where $\mathbf{conGs}(C, j_{t''}, \widetilde{io_{t''}})$.

On the other hand according to the definition of T, for all I/O-histories $t \notin T$ there is no associating clause in $P_A \cup P_B$. Consequently $\bar{t}$ constructed from t does not have the associating clause in $P_A \cup P_B$. This means any I/O-historie satisfying condition (5) is never in $\mathcal{O}_\Omega(P_A \cup P_B)$. Thus we proved $\mathcal{D}_y \not\subseteq \mathcal{O}_\Omega(P_A \cup P_B)$ for all y. Consequently

$$\mathcal{O}_\Omega(P_A \cup P_B) \subseteq \mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B).$$

Theorem 4.3 For any GHC Ω open programs P_A and P_B,

$$\mathcal{O}_\Omega(P_A \cup P_B) = \mathcal{O}_\Omega(P_A) \bigcup_\Omega \mathcal{O}_\Omega(P_B).$$

Proof: Obvious from Lemma 4.1 and Lemma 4.2.

5. Application to CC

We constructed an OR-compositional semantics for GHC programs. But the approach adopted in this paper can be applied not only to GHC but also to other concurrent logic lan-

guages. For example we can construct an OR-compositional semantics for CC programs in a similar way.

An unfolding semantics is presented in Ref. 5). The semantics of a cc($\mathcal{C}$) program is a set of reactive behaviors which are trees abstractly representing all the possible computations of a program. Given a clause C of a cc($\mathcal{C}$) program, A reactive behavior associated with C is obtained by applying *Unfolding rules* to C. This semantics is not OR-compositional as shown by next example.

Example 5.1 *Let P_A, P_B, P_C be cc($\mathcal{C}$) programs as follows:*

$$P_A = \{p(X,Y) :- X = b : \tau \rightarrow q(Y).\ q(Y) :- \tau : Y = a \rightarrow nil.\}$$
$$P_B = \{p(X,Y) :- X = b : \tau \rightarrow \tau : Y = a \rightarrow nil.\ q(Y) :- \tau : Y = a \rightarrow nil.\}$$
$$P_C = \{q(Y) :- \tau : Y = b.\}$$
$$P_A \cup P_C = \{p(X,Y) :- X = b : \tau \rightarrow q(Y).\ q(Y) :- \tau : Y = a \rightarrow nil + \tau : Y = b \rightarrow nil.\}$$
$$P_B \cup P_C = \{p(X,Y) :- X = b : \tau \rightarrow \tau : Y = a \rightarrow nil.\ q(Y) :- \tau : Y = a \rightarrow nil + \tau : Y = b \rightarrow nil.\}$$

The unfolding semantics of each program is as follows:

$$\mathcal{UNF}(P_A) = \mathcal{UNF}(P_B) = \{p(X,Y) :- X = b : \tau \rightarrow \tau : Y = a \rightarrow nil.\ q(Y) :- \tau : Y = a \rightarrow nil.\}$$
$$\mathcal{UNF}(P_C) = \{q(Y) :- \tau : Y = b.\}$$
$$\mathcal{UNF}(P_A \cup P_C) = \{p(X,Y) :- X = b : \tau \rightarrow \tau : Y = a \rightarrow nil + \tau : Y = b \rightarrow nil.\ q(Y) :- \tau : Y = a \rightarrow nil + \tau : Y = b \rightarrow nil.\}$$
$$\mathcal{UNF}(P_B \cup P_C) = \{p(X,Y) :- X = b : \tau \rightarrow \tau : Y = a \rightarrow nil.\ q(Y) :- \tau : Y = a \rightarrow nil + \tau : Y = b \rightarrow nil.\}$$

Clearly

$$\mathcal{UNF}(P_A \cup P_C) \neq \mathcal{UNF}(P_B \cup P_C). \quad (6)$$

If there exists OR-composition operator $\cup_*$ on semantic domain then

$$\mathcal{UNF}(P_A) \cup_* \mathcal{UNF}(P_C) = \mathcal{UNF}(P_A \cup P_C).$$
$$\mathcal{UNF}(P_B) \cup_* \mathcal{UNF}(P_C) = \mathcal{UNF}(P_B \cup P_C).$$

As $\mathcal{UNF}(P_A) = \mathcal{UNF}(P_B)$, left-hand-side of (5.1) *= left-hand-side of* (5.1). *So right-hand-side of each equation must be equal. This contradicts with* (6).

In order to distinguish $\mathcal{UNF}(P_A)$ with $\mathcal{UNF}(P_B)$ and make this semantics OR-compositional, we extend unfolding rules and construct OR-composition operation.

5.1 Extended Unfolding Rules

Unfolding rules in Ref. 5) consist of three rules.

i. Replacement of procedure calls by procedure definitions.
ii. Transformation of AND nodes to OR nodes.
iii. Freezing of failed and deadlocked paths.

We extend rule i. as follows. Rule ii. and iii. are available as they are.

Definition 5.2 (Extended rule i.) Let P be a cc($\mathcal{C}$) program and C be a clause in P. Every procedure calls p appearing in C are replaced by its definition in P if $pred(p) \not\subseteq \Omega$, otherwise p is replaced by or-tree such as

$p +$ (definition of p).

Unfolding semantics using extended rule i. can distinguish $P_A with P_B$ since

$$\mathcal{UNF}(P_A) = \{p(X,Y) :- X = b : \tau \rightarrow \tau : Y = a \rightarrow nil + q(Y).\ q(Y) :- \tau : Y = a \rightarrow nil.\}$$
$$\neq \mathcal{UNF}(P_B) = \{p(X,Y) :- X = b : \tau \rightarrow \tau : Y = a \rightarrow nil.\ q(Y) :- \tau : Y = a \rightarrow nil.\}$$

So the contradiction which occurred in a previous example has disappeared .

Formal OR-compositional semantics of cc($\mathcal{C}$) using this idea and construction of OR-composition operation will be shown in our future work.

6. Conclusion

This paper presented an OR-compositional semantics for GHC programs with perpetual processes by extending the semantics constructed in Ref. 11) and using the idea of Ref. 2). Though our semantics is OR-compositional and can express perpetual processes, it cannot distinguish processes which may deadlock from processes which does not deadlock. So we must extend this semantics to be deadlock sensitive by adding more information to I/O-history. For example I/O-history with tree structure is one approach.

References

1) Beckman, L.: Towards a Formal Semantics for Concurrent Logic Programming, *Proc. Third Int. Conf. Logic Programming*, pp.335–349 (1986).
2) Bossi, A., Gabbrielli, M., Levi, G. and Meo,

M.C.: Contributions to the Semantics of Open Logic Prgrams, *Proc. the International Conference on Fifth Generation Computer Systems*, pp.570–580 (1992).
3) Falaschi, M., Levi, G., Martelli, M. and Palamodessi, C.: A New Declarative Semantics for Logic Languages, *Proc. 5th Conf. and Symp.*, pp.993–1005 (1988).
4) Gabbrielli, M. and Levi, G.: Unfolding Reactive Semantics for Concurrent Constraint Programs, *Lecture Notes in Computer Science*, Vol.463, pp.204–216 (1990).
5) Gabbrielli, M. and Levi, G.: Unfolding and Fixpoint Semantics of Concurrent Constraint Logic Programs, *Theoretical Computer Science*, Vol.105, pp.85–128 (1992).
6) Gaifman, H., Maher, M.J. and Shapiro, E.: Reactive Behavior Semantics for Concurrent Constraint Logic Programs, *Proc. North American Conf. on Logic Programming*, pp.553–569 (1989).
7) Gerth, R., Codish, M., Lichtenstein, Y. and Shapiro, E.: Fully Abstract Denotational Semantics for Flat Concurrent Prolog, *Proc. North American Conf. on Logic Programming*, pp.553–569 (1989).
8) Levi, G. and Palamidessi, C.: Contributions to the Semantics of Logic Perpetual Processes, *Acta Informatica 25*, pp.691–711 (1988).
9) Lloyd, J.W.: *Foundations of Logic Programming*, Springer-Verlag (1984).
10) Maher, M.: Logic Semantics for a Class of Commited Choice Programming, *Proc. Forth Int. Conf. on Logic Programming*, The MIT Press, pp.858–876 (1987).
11) Murakami, M.: A Declarative Semantics of Flat Guarded Horn Clauses for Programs with Perpetual Processes, *Theoretical Computer Science*, Vol.75, pp.67–83 (1990).
12) Ueda, K.: Guarded Horn Clauses, Technical Report, TR-103, ICOT (1985).

(Received January 5, 1995)
(Accepted April 12, 1996)

Toru Kato is a student of Graduate School of Natural Science and Technology of Okayama University. His current research interests is formal semantics of concurrent logic programming languages. He graduated Graduate School of Engneering of Okayama University in 1993.

Masaki Murakami is an associate professor of Division of Logic and Computation in Department of Information Technology of Okayama University since 1991. His current research interests are the theory of concurrency, formal semantics of programming languages and program transformation of concurrent programs. He graduated Department of Information Engneering of Nagoya Institute of Technology in 1980. He received his Dr. of Engineering from Nagoya University. He joined Fifth Generation Computer Project from 1985 to 1989. He was working for Fujitsu from 1989 to 1991.