ソフトウェア基礎論　38－2
プログラミング言語　28－2
（１９９１．３．８）

メタと Reflective GHC における操作的意味論

沈 涵　　田中 二郎

富士通研究所（株）国際情報社会科学研究所

並列論理型言語に関する基礎的な研究、特にそのフォーマルな表現や意味論についての研究は重要である。本論文では、メタと reflection 機能をもつ並列論理型言語に対して操作的意味論を提案する。

まず、メタと reflection を表すためのフォーマルな表現の枠組を定義する。その表現の枠組によって表されたペアの集合に対して、一般的な unification アルゴリズムを定義する。さらに、ラベルをもつ transition relation に基づくメタと Reflective GHC における操作的意味論を提案する。

An Operational Semantics for Meta and Reflective GHC

Han SHEN　　Jiro TANAKA

International Institute for Advanced Study of Social Information Science,
FUJITSU LABORATORIES LIMITED

17-25, SHINKAMATA, 1-CHOME,OTA-KU, TOKYO 144, JAPAN

It is vital importance to take a first step towards the development of a theoretical basis dealing with not only concurrent logic programming but also the meta and reflective issues, especially in some important formal representation and semantic aspects.

Based on the work on operational semantics for full GHC, this paper proposes an operational semantics for meta and reflective GHC. After giving a formal representation framework, we define a general unification algorithm which copes with meta and reflective representation and GHC head unification in a uniform way. By using the algorithm, we propose an operational semantics based on labeled transition relation of states for meta and reflective GHC.

1 Introduction

Meta and reflective concurrent logic programming is of vital importance and usefulness. A meta program is a program which uses another program (lower level program) as data. Meta-programming techniques underlie many of the applications of logic programming. Reflective predicates implemented using meta-programming facilities have the capabilities of catching the current state of a system and modifying it dynamically.

As pointed out in [Hill 88]: "In spite of the fact that meta-programming techniques are widely and successfully used, the foundations of meta-programming and meta-programming facilities provided by currently available Prolog systems are by no means satisfactory." For sequential logic programming, a few researches have been done on the theoretical foundations of meta and reflective logic programming so far (investigating the semantics of the extensions to classical logic programming languages)[Hill 88][Subrahmanian 88][Lloyd 88] [Suga 90]. As for the semantics of concurrent logic programming languages, several approaches have been proposed [Ueda 90] [Levi 87] [Murakami 90a] [Gerth 88] [Murakami 90b][Maher 87], but the meta and reflective issues have not been addressed. On the other hand, though [Tanaka 88][Tanaka 90] have shown the relations of meta-interpreters and reflective operations for GHC, the main concern was only in implementation and application aspects. Therefore, it is extremely important to take a first step towards the development of a theoretical basis dealing with not only concurrent logic programming but also the meta and reflective issues, especially in some important formal representation and semantic aspects.

Based on the work on operational semantics for full GHC[Shen 90], the current paper proposes an operational semantics for meta and reflective GHC. In the following section, we give a formal representation framework for meta and reflective GHC, which is critical for defining satisfactory semantics. Then we describe a general unification algorithm which can deal with meta, reflective representation and GHC head unification constraint in a uniform way in section 3. In section 4, we propose an operational semantics based on labeled transition relation of states for meta and reflective GHC.

The paper assumes a basic knowledge of concurrent logic programming language GHC[Ueda 85] and some semantics aspects of Prolog[Lloyd 87].

2 A Formal Representation Framework for Meta and Reflective GHC

In this section, we give a uniform representation framework for meta and reflective GHC formally. One important design philosophy of our representation framework is to introduce *meta level* information into variables, predicates and program clauses so that they can be dealt with in a uniform way.

Let L be an object language whose syntax is the same as that of GHC[Ueda 85]. Let L' be a meta and reflective language which contains all the symbols in L, together with additional symbols such as "!", "$\lfloor$", "$\rfloor$", "$\lceil$", "$\rceil$" used for meta representations of variables, clauses, and primitive term-pair sets (they are called special constants which will be elaborated on in this section).

In order to distinguish with *special constants* such as variable representations, clause constants and primitive term-pair set constants (in language L'), constants which can not be treated furtherly are called *primitive constants.*

We use $VARS$ to denote all the variables originally appearing in a system.

In addition to the variables belonging to $VARS$, we also need the notion of *variant form variables* for defining operational semantics (refer to [Shen 90]). Intuitively, a variant form variable is a kind of *'locked'* variable which is constituted by marking a variable in $VARS$ with one or several "$*$", and the variable in $VARS$ is called the *original form variable.* For example, let $x \in VARS$, x^*, x^{**}, x^{***}, ... are the variant form variables of x, and their original form variable is x. A set of all the variant form variables with their original form variables in $VARS$ is denoted by $VARS^*$.

Furtherly, the predicate symbols occurring in a system are partitioned into three sets:

P_S, which contains all the system-defined predicate symbols;
P_R, which contains all the reflective predicate symbols;
$P_{R'}$, which contains all the check-only re-

flective predicate symbols $P_{R'} \subseteq P_R$ [1];
P_O, which contains all the ordinary predicate symbols.

We stipulate $PRED =_{df} P_S \cup P_R \cup P_O$, which contains all the predicate symbols appearing in a system, and $PRED =_{df} P_S \cup P_{R'} \cup P_O$, which is a subset of $PRED$.

Now, we define variable representations at meta levels:

Definition 1 (Variable Representation)
!X is said to be the variable representation of X at one level higher, where

(1) X is a variable at one level lower or;

(2) X is the variable representation of a variable at lower level. ■

Next, we define terms in L' inductively:

Definition 2 (Term in Language L')
A term in language L' is defined as follows:

(1) a is a term, where a is a primitive constant;

(2) a variable representation is a term;

(3) $\lfloor p(t_1 ... t_s), q_1(u_{11} ... u_{1f_1}), ..., q_m(u_{m1} ... u_{mf_m}), |, r_1(v_{11} ... v_{1h_1}), ..., r_n(v_{n1} ... v_{nh_n}) \rfloor$ *is a term, where* $p \in (P_O \cup P_R)$, $q_i \in PRED'$, $r_k \in PRED$ *and* t_l, u_{ij}, v_{kw} *are terms containing no variables in* $(VARS \cup VARS^*)$[2] $(i = 0, 1, ..., m,\ k = 0, 1, ..., n,\ l = 1, ..., s,\ j = 0, 1, ..., f_i,\ w = 0, 1, ..., h_k)$;

(4) $\lceil [t_1, X_1], \ ..., [t_m, X_m] \rceil$ *is a term, where* X_i *are variable representations and* t_i *are terms containing no variables in* $(VARS \cup VARS^*)$ $(i = 0, 1, ..., m)$;

(5) x is a term, where $x \in (VARS \cup VARS^*)$;

(6) $f(t_1 \ ... \ t_n)$ *is a term, where f is an n-ary function symbol and* t_i *are terms* $(i = 1, ..., n)$. ■

In the above definition, (1) says that a term of primitive constant is still a term in language L'. (2) says that a variable representation, i.e., a variable preceded by several ! is a term. A term constructed by (3) corresponds to a GHC clause at lower level, and the case of $i = 0, j = 0, k = 0, w = 0$ intends to express a unit clause. A clause at lower level is represented by a special constant indicated by $\lfloor \ ... \ \rfloor$, which is allowed further treatment. A term corresponds to (4) is the meta representation for a primitive term-pair set at lower level. Differing from primitive constants, it is quite natural to view the terms resulting from (2), (3), (4) as *special constants* which can be dealt with at higher meta levels. Actually, there are two types of constants in our representation framework. One is the type of primitive constants, which can not be treated furtherly. Another is the type of special constants which are higher level meta representations for variables, clauses, and primitive term-pair sets of lower levels. Furtherly, (5), (6) just mean that a term can be constructed in the same way as in pure logic programming.

The terms constructed in the above way are classified into a set $TERMS^*$ which consists of terms containing $x \in VARS^*$, and a set $TERMS$ which consists of terms containing no $x \in VARS^*$.

For convenience, we use $\hat{t}$ to stand for an n-ary term-tuple $t_1, ..., t_n$, and $p(\hat{t})$ to stand for an n-ary literal, here t_i $(i = 1, .., n)$ are terms and p is a predicate symbol. A set consisting of all the literals is denoted by LI. Moreover, $t \equiv s$ means that term t is syntactically equal to term s.

Definition 3 (Clause)
Let $q_i \in PRED'$, $r_j \in PRED$, $\hat{t}$, $\hat{t_i}$,$\hat{s_j}$ *represent term-tuples* $(i = 1, ..., m,\ j = 1, ..., n)$, *Env, Nenv represent current and new environments respectively.*[3] *A clause C in L' has either the form:*

$$p(\hat{t}) \leftarrow q_1(\hat{t_1}), ..., q_m(\hat{t_m}) | r_1(\hat{s_1}), ..., r_n(\hat{s_n})$$

, which is called an ordinary clause, *where* $p \in P_O$; *or the following form:*

$$reflect(p(\hat{t})\ Env\ Nenv) \leftarrow q_1(\hat{t_1}), ..., q_m(\hat{t_m}) | r_1(\hat{s_1}), ..., r_n(\hat{s_n})$$

, which is called a reflective clause, *where* $p \in P_R$. *Here* $p(\hat{t})$ *is called the* head of C, *the conjunction of the literals* $q_1(\hat{t_1}), ..., q_m(\hat{t_m})$ *is called the* guard goals of C, *and the conjunction of the literals* $r_1(\hat{s_1}), ..., r_n(\hat{s_n})$ *is called the* body goals of C. *The operator "|" is called the commit operator.* ■

[1] check-only reflective predicates are those only check current program environment and do not modify it

[2] Actually, these terms are constructed from primitive constants, special constants and functors

[3] for details, see section 4

This paper assumes that variable sets occurring in different clauses are disjoint.

A meta and reflective GHC program is defined as follows:

Definition 4 (Program)
A finite set of ordinary and reflective clauses is referred to be as a meta and reflective GHC program. ■

Finally, a goal in L' is defined as follows:

Definition 5 (Goal)

$$\leftarrow p_1(\hat{t}_1), ..., p_m(\hat{t}_m)$$

, *where* $p_i \in PRED$, $\hat{t}_i$ *are term-tuples* $(i = 1, ..., m)$. ■

The above notions are different from the corresponding ones[Ueda 85] in the following aspects:

(1) our notions extend the concept of variables into a generalized one involving both variables and variant form variables;

(2) instead of only one type of primitive constants, our term definition also allows special constants which are higher level meta representations for variables, clauses, and primitive term-pair sets of lower levels.

Now, we define a lift function Ψ, which transforms program clauses (including unit clauses) into meta-level representations at higher levels.

Definition 6 (Lift Function Ψ)

(1) $\Psi(c) =_{df} c$, *where c is a primitive constant;*

(2) $\Psi(X) =_{df} !X$, *where X is a variable* $x \in (VARS \cup VARS^*)$ *or variable representation at meta level;*

(3) $\Psi(f(t_1 \ ... \ t_n)) =_{df} f(\Psi(t_1) \ ... \ \Psi(t_n))$, *where f is an n-ary function symbol and* $t_i \in (TERMS \cup TERMS^*)$ $(i = 1, ..., n)$;

(4) $\Psi(p(t_1 \ ... \ t_n)) =_{df} p(\Psi(t_1) \ ... \ \Psi(t_n))$, *where* $p \in PRED$ *and* $t_i \in (TERMS \cup TERMS^*)$ $(i = 1, ..., n)$;

(5) $\Psi(L_1, ..., L_n) =_{df} \Psi(L_1), ..., \Psi(L_n)$, *where* $L_i \in LI$ $(i = 1, ..., n)$;

(6) $\Psi(H \leftarrow G_1, ..., G_n | B_1, ..., B_m) =_{df}$ $\lfloor \Psi(H), \Psi(G_1), ..., \Psi(G_n), |, \Psi(B_1), ..., \Psi(B_m) \rfloor$, *where* $H, G_i, B_j \in LI$ $(i = 0, 1, ..., n, \ j = 0, 1, ..., m)$ [4];

(7) $\Psi(\lfloor H, G_1, ..., G_n, |, B_1, ..., B_m \rfloor)$ $=_{df} \lfloor \Psi(H), \Psi(G_1), ..., \Psi(G_n), |, \Psi(B_1), ..., \Psi(B_m) \rfloor$, *where* $H, G_i, B_j \in LI$ $(i = 0, 1, ..., n, \ j = 0, 1, ...m)$. ■

(1) says we need not to mark a primitive constant at higer level. (2)(3)(4)(5) say that the function Ψ transforms variables, variable representations, functors, literals, and conjunctions of literals into corresponding meta representations at one level higer. In (6), a clause at one level lower is indicated by $\lfloor \ ... \ \rfloor$, which is a *special constant* to be allowed further treatment. (7) just says that it has the property of idempotent.

By using the function Ψ, we can easily define a lift function Φ which transforms term-pair sets into meta representations at higher levels:

Definition 7 (Lift Function Φ)

(1) $\Phi(\{< t_1, x_1 >, ..., < t_m, x_m >\}) =_{df}$ $\lceil [\Psi(t_1), \Psi(x_1)], \ ... \ , [\Psi(t_m), \Psi(x_m)] \rceil$, *where* $x_i \in (VARS \cup VARS^*)$, $t_i \in (TERMS \cup TERMS^*)$ $(i = 1, ..., m)$;

(2) $\Phi(\lceil [t_1, X_1], \ ... \ , [t_m, X_m] \rceil) =_{df}$ $\lceil [\Psi(t_1), \Psi(X_1)], \ ... \ , [\Psi(t_m), \Psi(X_m)] \rceil$, *where* X_i *are variable representations and* t_i *are terms containing no variables in* $VARS \cup VARS^*$ $(i = 1, ..., m)$. ■

We also assume an inverse function Ψ^{-1} which transforms meta representations into the corresponding clauses or goals at lower levels, and an inverse function Φ^{-1} which transforms meta representations into the corresponding primitive term-pair sets at lower levels. Their definitions are omitted in the paper.

Now, we state a little more about our representation framework:

Our representation framework distinguishes primitive constants from those special constants (in language L') which are meta representations for variables, clauses, and primitive term-pair sets of lower levels.

[4] $i = 0$, $j = 0$ is used to express the case of unit clause

A variable (in $VARS \cup VARS^*$) at lower level is marked by the symbol !. Number of ! which occurs ahead of it represents what meta level it is currently 'lifted' to. Such a variable representation is just viewed as a *special constant* by current program. It is not a variable in current program any more. For example, $!!!x$ means that the variable x is now 'lifted' into the meta level 3 since there exists three ! ahead of x. It is a special constant at current meta level 3. Furthermore, the level of a variable occurring in current program is the same as that of the term bound to it. Therefore, our variable representation is a kind of *ground representation* [Hill 88] since it represents a variable at lower level (may be a meta level variable) by a special constant (preceded by one or several '!') at higher meta level.

In our representation framework, every predicate is assumed to have corresponding levels. Differing from only two levels of one object level and one meta level in [Hill 88], our framework has multi-level meta representation capability. A current (meta-level) program views lower level variables, program clauses, and primitive term-pair sets as *special constants* with their level information indicated by number of ! and the special symbols like $\lfloor$, $\rfloor$, $\lceil$, $\rceil$. Our representation framework can be viewed as a generalized ground representation, rather than pure typed representation or ground representation which deal with only two levels[Hill 88]. It is a kind of ground representation because it represents a variable by a special constant at higher level. And it not merely represents lower level variables by special constants, but also represents lower level clauses and goals by special constants with meta level information as well. In fact, it can represent multi meta-level information (theoretically, infinite reflective tower), which is impossible in pure typed representation or ground representation with only two types of "o" and "u" [Hill 88]. Furthermore, our representation framework allows special constants to be treated in a uniform way (such as in the general unification algorithm) without extra type information.

3 A General Unification Algorithm For Meta and Reflective GHC

Based on our meta representation framework, this section generalizes the extended unification algorithm for full GHC[Shen 90] into a general unification algorithm which can deal with meta, reflective representation and GHC head unification constraint in a uniform way.

The notions of *pair set*, *primitive term-pair set*, *solved substitution*, *suspended substitution*, and the operations of *plus-operation* and *minus-operation* defined in [Shen 90] can be easily extended into the corresponding ones by using the generalized version of term definition in which involves both primitive constants and special constants and allowing predicates to include reflective ones. The revised versions are omitted in this paper.

Other concepts such as most general unifier (mgu), renaming equivalence and answer substitution are quite similar to those in logic programming [Lass 88], we also omit their definitions in this paper.

Now, we give a general unification algorithm for the computation of a most general unifier of a given pair set. It is a generalized version of the extended unification algorithm for full GHC[Shen 90]. Here we use θ to denote a solved substitution and $\theta^{(s)}$ to annotate a suspended substitution.

General Unification Algorithm

Let S be a pair set. Repeatedly choose a pair in S of the following form and perform the corresponding action until it terminates with failure or nothing can be done for S furtherly.

(1) $< x, x >$ (or $< c, c >$), where $x \in (VARS \cup VARS^*)$ (or c is a constant):
delete the pair from S;

(2) $< c_1, c_2 >$, where at least one of c_1, c_2 is a primitive constant and $c_1 \not\equiv c_2$:
terminate with failure;

(3) $< c_1, x >$, where $x \in VARS^*$ and c_1 is a constant (primitive or special constant):
if there exists another pair $< c_2, x >$ in S (c_2 is a constant) and $c_1 \equiv c_2$ then delete $< c_2, x >$ from S;

(4) $< f(u_1 \ldots u_m),\ g(v_1 \ldots v_n) >$, where f, g are function symbols and $u_i, v_j \in (TERMS^* \cup TERMS)$ $(i = 1, ..., m,\ j = 1, ..., n)$:
if $f = g$ and $m = n$ then replace it by the pairs: $< u_1, v_1 >$,..., $< u_m, v_m >$ else terminate with failure;

(5) $< p(u_1 \ldots u_m),\ q(v_1 \ldots v_n) >$, where $p, q \in (P_O \cup P_R)$ and $u_i, v_j \in (TERMS^* \cup TERMS)$

$(i = 1, ..., m,\ j = 1, ..., n)$:
if $p = q$ and $m = n$ then replace it by the pairs $< u_1, v_1 >, ..., < u_m, v_m >$ else terminate with failure;

(6) $< x,\ t >$, where $x \in (VARS^* \cup VARS)$, $t \in (TERMS^* \cup TERMS) \wedge t \notin VARS$, $x \notin VARS^* \vee t \notin VARS^*$:
substitute the pair by $< t, x >$;

(7) $< t,\ x >$, where $x \in VARS$ and x occurs in the other pairs of S, $t \in (TERMS^* \cup TERMS)$:
if $x \in t$ then terminate with failure else replace x in the other pairs of S by t;

(8) $< X,\ t >$ (or $< t,\ X >$), where X is a variable representation and $t \in (TERMS^* \cup TERMS) \wedge t \notin (VARS^* \cup VARS)$:
replace it by the pair $< t,\ X >$ and replace all the X in the other pairs of S by t;

(9) $< t,\ \lfloor H, G_1, ..., G_m, |, B_1, ..., B_n \rfloor >$, (or $< \lfloor H, G_1, ..., G_m, |, B_1, ..., B_n \rfloor,\ t >$), where H, G_i, B_j $(i = 0, 1..., m,\ j = 0, 1, ..., n)$ are literals, and t is a list or the meta representation of a clause at lower level[5]:
replace it by the pair:
$< t,\ [H, G_1, ..., G_m, |, B_1, ..., B_n] >$;

(10) $< t,\ \lceil [t_1, X_1], ..., [t_m, X_m] \rceil >$, (or $< \lceil [t_1, X_1], ..., [t_m, X_m] \rceil,\ t >$), where t_i are terms containing no variables in $VARS \cup VARS^*$ and X_i are variable representations $(i = 1, ..., m)$, and t is a list or the meta representation of a primitive term-pair set at lower level[6]:
replace it by the pair:
$< t, [[t_1, X_1], ..., [t_m, X_m]] >$.

Finally, if there exists $< t, x > \in \theta$, where $x \in VARS^*$, $t \in (TERMS^* \cup TERMS)$; then rename S by $\theta^{(s)}$ (suspended substitution) else rename S by θ. ■

Notice that terms appearing in the above algorithm can contain meta level information.

Compared with the extended unification algorithm [Shen 90], this algorithm deliberately treats constants involving primitive constants and special constants. In the action(7), variable x (may be a meta level variable) never occurs in a form of meta representation in the pairs of S, due to our assumption that variables in different clauses should be disjoint, in other words, there should be no x's meta representation in S. So it causes no problem in meta GHC. Furtherly, the action (8) is added to cope with the unification for meta representations of lower level variables. Variables at lower level (i.e., variable representations at current level) are allowed to be changed at current level in order to implement reflective capability. The action (9) deals with the unification with special constants of meta representations for clauses at lower levels. The meta representation of a clause at lower level is revised as a list if the other side (t) of a pair is a list or the meta representation of a clause at lower level so that the action (4) can be applied for further unification. Similarly, the action (10) applies to the case of meta representations for primitive term-pair sets at lower levels.

The following procedure shows how the above general unification algorithm is used in our computing procedure of operational semantics:

Main Procedure of Unification

- For a goal of the form: $G_1 = G_2$:
  create a pair set $S = \{< G_1, G_2 >\}$, and then apply the general unification algorithm to it;
- For a system-defined computing goal G:
  firstly compute G, and then create a primitive term-pair set whose elements have the form of $< c, x >$, where x is a variable in G, $x \in (VARS \cup VARS^*)$ and c is a primitive constant which is the result of x by computing G;
- For an ordinary or reflective goal G:
  firstly apply the plus-operation on G, then make a pair set S: $\{<^+ (G),\ H >\}$, here H is the head of a clause C which is tried to be unified with G (in the case of a reflective goal, C has the form: $reflect(H\ Env\ Nenv) \leftarrow Guard\ |Body$). Furtherly, apply the general unification algorithm to the pair set S. ■

It holds clearly that:

Proposition 1

For a given pair set S, the general unification algorithm can obtain a solved substitution which is the mgu of S, or a suspended substitution towards the mgu of S, or terminates with failure in a finite time. ■

[5] i.e., t has the form of [...] or $\lfloor, ..., \rfloor$
[6] i.e., t has the form of [...] or $\lceil, ..., \rceil$

We also need a concatenation rule ($\theta_1 \circ \theta_2$) which is used to concatenate two primitive term-pair sets θ_1, θ_2 (exactly, θ_1 is a solved substitution and θ_2 is a primitive term-pair set) in the computing process of our operational semantics. It is a little different from the one for full GHC[Shen 90] in that a binding (a primitive term-pair) in θ_2 may replace the old one in θ_1 just because a reflective goal corresponding to θ_2 can change a binding generated before (in θ_1).

4 An Operational Semantics For Meta and Reflective GHC

By using the general unification algorithm defined in the previous section, we propose an operational semantics for meta and reflective GHC based on labeled transition relation of states.

Thanks to our notion of *variant form variable*, *meta representation framework*, the notions of *suspended substitution* and *solved substitution*, the operations of *plus-operation* and *minus-operation*, the *general unification algorithm* , and the *concatenation rule*, we can define a simple and intuitive operational semantics for meta and reflective GHC.

First, we need the following definition of restricted primitive term-pair set:

Definition 8

Let S be a primitive term-pair set, G be a goal, and $V(G)$ ($\subseteq VARS$) be the set of all the variables appearing in G.

Restrict S to G is defined as:
$S|_G =_{df} \{< t, x > \mid < t, x > \in S \wedge x \in V(G)\}$. ■

Now, we define the notion of *labeled transition relation of states* upon which our operational semantics is based.

Assume $GOAL$ represent the set of all the goals, PRO represent the set of all the meta and reflective GHC programs, DA represent the set of meta representation of all the meta and reflective GHC programs, and SUB represent the set of all the primitive term-pair sets. We express a *state* by an element in $STATE = GOAL * PRO * SUB * DA$.

Definition 9 (Labeled Transition Relation)

Let $s_i, s_k \in STATE$. $s_i = [G_i, P_i, \theta_i, D_i]$, where $label(G_i) = l$. A labeled transition relation on states is a relation satisfying the following conditions:

(1) G_i is a single goal:

(1a) $s_i \xrightarrow{l} s_k$ holds, where $s_k \equiv s_i$;

(1b) $G_i \equiv G_{i1} = G_{i2}$:
if the general unification algorithm obtains a primitive term-pair set σ for the pair set: $\{< G_{i1}, G_{i2} >\}$, then $s_i \xrightarrow{l} s_k$ holds, where $s_k = [null, P_i, \theta_i \circ \sigma, D_i]$;

(1c) $G_i \equiv q(\hat{t})$ and q is a system-defined computing predicate:
if the general unification algorithm obtains a primitive term-pair set σ corresponding to the computing result of G_i[7], then $s_i \xrightarrow{l} s_k$ holds, where $s_k = [null, P_i, \theta_i \circ \sigma, D_i]$;

(1d) $G_i \equiv q(\hat{t})$ and $q \in P_O$ (ordinary predicate):
if the general unification algorithm yields a suspended substitution σ for the pair set: $\{<^+ (G_i), H >\}$, where H is the head of a clause C: $H \leftarrow Guard \mid Body$ ($C \in P_i$), then $s_i \xrightarrow{l} s_k$ holds, where $s_k = [G_i, P_i, \theta_i \circ \sigma, D_i]$;
else if exists a solved substitution σ for the pair set: $\{<^+ (G_i), H >\}$, where H is the head of a clause C: $H \leftarrow Guard \mid Body$ ($C \in P_i$) and $[(Guard)\sigma, P_i, \theta_i \circ \sigma, D_i] \xrightarrow{l} \ldots \xrightarrow{l} [null, P_i, \theta_i \circ \sigma \circ \gamma, D_i]$, then $s_i \xrightarrow{l} s_k$ holds, where $s_k = [^-((Body)(\sigma \circ \gamma)), P_i, \theta_i \circ \sigma \circ \gamma, D_i]$

(2) G_i is composed of several subgoals, suppose, $G_i \equiv G_{i_1}, \ldots, G_{i_m}$. We assume at most one reflective subgoal can have a real transition relation while all the other subgoals only have reflexive one:

(2a) If $G_{i_u} \equiv q(\hat{t})$ ($1 \leq u \leq m$) is a reflective subgoal in G_i ($q \in P_R$):
applying the lift function Ψ to G_{i_u}, get $\Psi(G_{i_u})$. if $\exists C : reflect(H\ Env\ Nenv) \leftarrow Guard \mid Body$ in Γ (Γ is a subprogram at meta level $l+1$), and the general unification algorithm obtains a solved substitution σ for the pair set: $\{<^+ (\Psi(G_{i_u})), H >\}$, set

[7] the elements of σ have the form of $< c, x >$, $x \in (VAR \cup VARS^*)$, c is a primitive constant

$D_i' = \Psi(P_i)$ *and the environment variable* $Env = [\Psi(P_i), \Phi(\theta_i), \Psi(G_i')]$, *where* $G_i' = G_{i_1}, ..., G_{i_{u-1}}, G_{i_{u+1}}, ..., G_{i_m}$, *furtherly, if* $[Guard\sigma, \Gamma, \sigma, D_i'] \xrightarrow{l+1} ... \xrightarrow{l+1} [null, \Gamma, \sigma \circ \gamma, D_i']$, *then* $s_i' \xrightarrow{l+1} s_k$ *holds, where* $s_i' = [\Psi(G_{i_u}), \Gamma, \varepsilon, D_i']$[8] *and* $s_k = [\neg((Body)(\sigma \circ \gamma)), \Gamma, \sigma \circ \gamma, D_i']$; *Furtherly, if* $s_k \xrightarrow{l+1} ... \xrightarrow{l+1} s_h$, *where* $s_h = [null, \Gamma', \sigma \circ \gamma \circ \zeta, D_i'']$ *and* $Nenv = [D_i'', \theta_z, G_z]$; *then* $s_i \xrightarrow{l} s_h'$ *holds, where* $s_h' = [\Psi^{-1}(G_z), \Psi^{-1}(D_i''), \Phi^{-1}(\theta_z), \Psi^{-1}(D_i'')]$;

(2b) if $[G_{i_j}, P_i, \theta_i, D_i] \xrightarrow{l} [Q_{i_j}, P_i, S_{i_j}, D_i]$ *(S_{i_j} are primitive term-pair sets, $j = 1, ..., m$) exists v such that* $[G_{i_v}, P_i, \theta_i, D_i] \not\equiv [Q_{i_v}, P_i, S_{i_v}, D_i]$ $(1 \le v \le m)$ *and no reflective subgoal has real transition relation, then* $s_i \xrightarrow{l} s_k$ *holds, where* $s_k = [(Q_{i_1}, ..., Q_{i_m})\Lambda(S_{i_1}|_{G_{i_1}}, ..., S_{i_m}|_{G_{i_m}}), P_i, \Lambda(S_{i_1}|_{G_{i_1}}, ..., S_{i_m}|_{G_{i_m}}), D_i]$, *and* $\Lambda(S_{i_1}|_{G_{i_1}}, ..., S_{i_m}|_{G_{i_m}})$ *is a solved substitution which is the result of the AND-combination of* $S_{i_1}|_{G_{i_1}}, ..., S_{i_m}|_{G_{i_m}}$. *It is defined as follows:*

Let S_i $(i = 1, ..., n)$ *be primitive term-pair sets, which correspond to subgoals* $G_1, ..., G_n$ *in a goal:* $\leftarrow G_1, ..., G_n$. *Initially, create a set:* $S = S_1 \cup ... \cup S_n$. *The AND-combination* $\Lambda(S_1, ..., S_n)$ *can be obtained by repeatedly performing the following actions:*

- *For each* $< t, x > \in S$ $x \in VARS$, $t \in (TERMS^* \cup TERMS)$, *replace all the variant form variables of* x *by* t;
- *Then apply the general unification algorithm for* S. ■

For a state with a single goal, (1a) says that our transition relation is a reflexive one. For the unify predicate "=", (1b) creats a pair set whose elements might contain meta level information and variant form variables, and applys the general unification algorithm to it. For a system-defined computing predicate such as *summary*, (1c) simply computes it and creates the corresponding primitive term-pair set. (1d) deals with user-defined ordinary predicates. If a suspension yields for the head unification, we adopt an active strategy which can characterize efficient implementation of GHC head unification and can also avoid deadlocks sometimes (see [Shen 90]). In the transition relation, the states s_i and s_k have the same goals and program, only have different primitive term-pair sets, the one in s_k is a suspended substitution. As for a successful head unification, our approach is the same as the one in [Shen 90].

[8] ε is a null primitive term-pair set

In (2), we make a constraint on computing the concurrent combination of subgoals in G_i, i.e., at most one reflective subgoal can have a *real* transition relation while all the other subgoals only have reflexive one. Because the execution of a reflective goal might catch and modify the current environment, so we assume that only one reflective subgoal G_{i_u} in G_i can go forward in one AND-combination computation of transition relations if the other subgoals of G_i have reflexive transition relation. The execution of several reflective subgoals in a goal can be characterized by computing AND-combination several times. For the reflective subgoal G_{i_u} (assuming at level l), transfer it, the current program and primitive term-pair set into the corresponding meta representations at level $l+1$. And set the environment variable ENV including the meta representations of current program, current primitive term-pair set and the remaining goals, and creat a state s_i' at level $l+1$, then establish some transition relations labeled as $l+1$. After the goals successful terminate at level $l+1$, do some level adjustments by applying the function Ψ^{-1} to the resulting program data and remaining goals data, and the function Φ^{-1} to the primitive term-pair set, creat a new state s_h' at level l, and establish the transition relation between the state s_i and s_h' labeled as l, i.e., $s_i \xrightarrow{l} s_h'$. Note that a transition relation holds only between the states at the same level. In the case (2a), the only transition relation that holds at label l is $s_i \xrightarrow{l} s_h'$, no transition relation holds between s_i and any of the states at level $l+1$. Furtherly, case (2b) describes the AND-combination for concurrent ordinary subgoals, which is similar to the one in [Shen 90].

Based on the above transition relation of states, we can define a simple operational semantics for meta and reflective GHC:

Definition 10 (Operational Semantics)
Let G, G_i $(i = 1, ..., m)$ *be goals,* P, P_i *be programs,* $D, D_i \in DA$, ε *be a null primitive term-pair set,* S_i $(i = 1, ..., m)$ *be primitive term-pair sets,* s *be a state:* $s = [G, P, \varepsilon, nill]$. *We call* $s = [G, P, \varepsilon, nill] \xrightarrow{l} [G_1, P_1, S_1, D_1] \xrightarrow{l} ... \xrightarrow{l}$

$[G_m, P_m, S_m, D_m]$ *an* execution *of* G. *Furtherly,*

- *If* $G_m = null$:
  then it is called a successful execution of G, *and* S_m *is referred to as an answer substitution of* G;
- *If* $G_m \neq null$,
  If S_m *is a suspended substitution, then it is called a suspended execution. Furtherly, if all the executions starting from goal* G *are suspended executions, we say the execution of* G *yields a deadlock;*

  else if there exists no transition relation for the state $[G_m, P_m, \theta_m, D_m]$ *furtherly, then it is called a failure execution.* ■

5 Final Remarks

It is vital importance to develop a theoretical basis dealing with not only concurrent logic programming but also the meta and reflective issues, especially in some important formal representation and semantic aspects. As a first step work, this paper proposes an operational semantics for meta and reflective GHC.

Firstly, a formal representation framework for meta and reflective GHC at higher meta-level has been defined, and then a generalized unification algorithm coping with meta, reflective facilities and GHC head unification has been described. Furtherly, an operational semantics based on labeled transition relation of states for meta and reflective GHC by using the general unification algorithm has been proposed.

It is clear that the meta and reflective problems can be traced to the fact that it doesn't handle the representation requirements properly. Once an appropriate representation is used, there is no theoretical impediments to obtaining a simple and satisfactory semantics. This paper has showed it by defining an operational semantics for meta and reflective GHC.

There remains much work to be done, which is related to the semantics of meta and reflective concurrent logic programming. We would try to characterize some issues such as declarative semantics for meta and reflective GHC. And prove some correctness of practical systems by use of the proposed semantics.

6 Acknowledge

The authors would like to express our thanks to Hiroyasu SUGANO for his joining to the discussions on the draft version of this paper. The research underlying this paper has benefited from his thoughtful comments and valuable advices. We would also like to thank to Masaki MURAKAMI for his discussions and comments.

References

[Clark 85] K. Clark and S. Gregory: PARLOG, Parallel Programming in Logic, Research Report DOC 84/4, Dept. of Computing, Imperial College of Science and Technology, Revised 1985.

[de Boer 89] Frank S. de Boer, Joost N.Kok, Catuscia Palamidessi and Jan J.M.M.Rutten: Semantic Models for a Version of PARLOG, Proceedings of the 6th International Conference on Logic Programming, pp. 621-636, J.-L.Lassez, Ed. MIT Press, Cambridge, Mass., 1989.

[Gerth 88] R. Gerth, M. Codish, Y. Lichtenstein and E. Shapiro: Fully Abstract Denotational Semantics for Flat Concurrent Prolog, Third Annual Symposium on Logic in Computer Science, pp. 320-335, Edinburgh, Scottland, IEEE, July 5-8, 1988.

[Hill 88] P.M. Hill and J.W. Lloyd: Analysis of Meta-Programs, Workshop on Meta-programming in Logic Programming (J.W. Lloyd ed.), pp.27-42, Bristol, June 1988.

[Lass 88] J.L. Lassez, M.J. Maher and K. Marriot: Unification Revisited, Foundations of Deductive Databases and Logic Programming (J.Minker,ed), pp.587-625, Morgan Kaufman Publishers, Los Altos, CA, 1988.

[Levi 87] G. Levi and C. Palamidessi: An Approach to the Declarative Semantics of Synchronization in Logic Languages, Logic Programming: Proc. of the Fourth International Conference, Vol. 2, pp. 877-893, The MIT Press, 1987.

[Lloyd 87] J.W. Lloyd: Foundations of Logic Programming (second edition), Springer-Verlag, 1987.

[Lloyd 88] Directions for Meta-programming, Proc. of the International Conference on Fifth

Generation Computer Systems, pp. 609-617, ICOT, 1988.

[Maher 87] M.J. Maher: Logic Semantics for a Class of Committed-Choice Programs, Proceedings of the 4th International Conference on Logic Programming, J.-L.Lassez, pp.858-876, Ed. MIT Press, Cambridge, Mass:, 1987.

[Murakami 90a] M. Murakami: A Declarative Semantics of Flat Guarded Horn Clauses for Programs with Perpetual Processes, Theoretical Computer Science 75, pp.67-83, North-Holland, 1990.

[Murakami 90b] M. Murakami: Formal Semantics of Concurrent logic Programs, Journal of Information Processing Society (to appear).

[Shapiro 83] E. Shapiro: A Subset of Concurrent Prolog and Its Interpreter, ICOT Technical Report, TR-003, 1983.

[Shapiro 89] E. Shapiro: The Family of Concurrent Logic Programming Languages, ACM Computing Surveys, pp.413-510, Vol.21, No.3 (1989).

[Shen 90] H. Shen and J.Tanaka: An Operational Semantics for Full GHC Based on Extended Unification Algorithm, COMP 90-66, pp.43-51, IPSJ,IEICE (1990).

[Subrahmanian 88] V.S. Subrahmanian: Foundations of Meta Logic Programming, Workshop on Meta-programming in Logic Programming (J.W. Lloyd ed.), pp.53-66, Bristol, June 1988.

[Suga 90] H.Sugano: Meta and Reflective Computation in Logic Programming, Workshop on Meta-Programming in Logic programming, pp.19-39, Leuven, Belgium, April 1990.

[Tanaka 88] J. Tanaka: Meta-interpreters and Reflective Operations in GHC, Proc. of International Conference on FGCS, pp.774-783, ICOT, 1988.

[Tanaka 90] J. Tanaka and F. Matono: Reflective GHC and Its Implementation, Proc. of the Logic Programming Conference'90, pp. 179-189, ICOT.

[Ueda 85] K. Ueda: Guarded Horn Clauses, ICOT Technical Report, TR-103, 1985.

[Ueda 90] K. Ueda: Designing Concurrent Programming Language, Proceedings of an International Conference organized by the IPSJ to Commemorate the 30th Anniversary, pp. 87-94, IPSJ 1990.