

TR-339

An Axiomatic Verification Method for
Synchronizations of Guarded Horn
Clauses Programs

by
M.Murakami

February, 1988

©1988, ICOT

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

An Axiomatic Verification Method for
Synchronizations of Guarded Horn Clauses Programs

Masaki Murakami

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

ABSTRACT: Guarded Horn Clauses (GHC) is a parallel programming language based on Horn logic. This paper proposes an axiomatic verification method for partial correctness of GHC program as Hoare logic. The system presented here can prove properties of the GHC program which are satisfied by synchronization mechanisms and cannot be proved by methods for pure Horn logic programs.

1. Introduction

During the last few years, several parallel programming languages based on Horn logic, such as PARLOG [Clark 86], Concurrent Prolog [Shapiro 86] and Guarded Horn Clauses (GHC) [Ueda 85] have been investigated. These languages are designed to represent the notions of processes and to provide mechanisms for communication and synchronization in a logic programming framework. In these languages, Horn logic is extended to describe these notions. In the case of GHC, a program consists of a finite set of Horn clauses with a commit operator, '|'.

Thus verification methods for pure Horn logic programs such as [Kanamori 86] are not enough to prove properties of programs which contain such synchronization operators. For example, Takeuchi [Takeuchi 86] introduced an example of a GHC program $\text{top}(X, Y)$. It satisfies the output condition $Y = [a, a]$ for the input condition $X = [a]$ by the control of synchronization mechanisms. It is impossible to show that top satisfies this specification by using verification methods for pure Horn logic programs. Thus the semantics of synchronizations are expected. Results on the formal semantics of parallel languages base on Horn logic have been reported in several sources [Ueda 86, Saraswat86, 87, Levi87, Takeuchi86, Maher87]. However, most of them are based on operational or fixedpoint approach. It is too complicated to apply these semantics to prove properties of given programs.

This paper adopts the axiomatic approach to give a logical framework as a verification method for the properties of GHC programs which are satisfied by synchronizations. A Hoare-like axiomatic system for proving the partial correctness of programs is modified and extended for GHC programs.

In this paper, several restrictions are assumed to GHC programs for the proof of properties. Most of them are for simplicity. Programs which do not satisfy the restrictions can be verified by a straightforward extension of the method presented in

this paper. However some of the restrictions are essential. One of the essential restriction is that the guards of clauses must be flat. However flat GHC is considered to be enough useful. Thus it is considered that there no problem to restrict the target of our verification method to flat GHC programs.

Another essential restriction is that data-dependencies in programs can be decided obviously. Namely it is assumed that for every occurrence of variables in the execution of the program, it can be decided whether it occurs as an input variable or an output, and for any variable which is shared between more than two processes, which process instantiates the variable can be decided uniquely. These conditions are assumed because that the dependencies are referred in the applications the inference rules of the system presented here. It is considered that the system presented here can be extended for the verification of programs that the dependencies of data cannot be decided obviously by introducing some annotations which denotes the dependencies that the programmer is conscious of implicitly. Thus it is considered that the method of verification presented here is not so rigid as the appearance.

2. Partial Correctness of GHC Programs

This section briefly introduces GHC and defines partial correctness for GHC.

2.1 Guarded Horn Clauses

Guarded Horn Clauses (GHC) is a parallel logic programming language. For a set of predicate symbols, PRED, function symbol, FUN, and variable symbol, VAR, a program of GHC consists of a finite set of guarded clauses. A guarded clause has the form:

$$H :- B_1, \dots, B_n \mid A_1, \dots, A_m.$$

where H is the head of the clause, H , B_1, \dots, B_n is the guard, and A_1, \dots, A_m is the body. Note that the clause head is included in the guard. Each $B_i (1 \leq i \leq n)$ has the form 'true' or $T = S$, where T and S are in the set of terms, TERM, constructed from FUN and VAR. Each $A_j (1 \leq j \leq m)$ takes the form $p(T_1, \dots, T_k)$ or $T = S$, where $p \in \text{PRED}$ and $T_i (1 \leq i \leq k) \in \text{TERM}$. H takes the form $p(T_1, \dots, T_k)$. The operator ' \mid ' is called the commitment operator. A goal clause takes the form of a body part and is denoted:

$$G_1, \dots, G_h$$

where each $G_i (1 \leq i \leq h)$ is called a goal. For the computation rule of GHC programs, see [Ueda 85]. The set of guarded clauses Db_a in following defines one of the programs which are called Brock-Ackermann's anomaly [Takeuchi 86].

$\text{Db}_a :$

$\text{top}(\text{In}, \text{Out}) :- \text{true} \mid \text{s}(\text{In}, \text{Mid}, \text{Out}), \text{plus1}(\text{Out}, \text{Mid}). \text{-----}(1)$

```

s(lx, ly, Out) :- true | dup(lx, Ox), dup(ly, Oy),
                    merge(Ox, Oy, Oz), pop(Oz, Out). -----(2)

pop([A, B|_], 0) :- true | 0 = [A, B]. -----(3)

dup([A|I], 0) :- true | 0 = [A, A]. -----(4)

merge([A|Ix], ly, 0) :- true | 0 = [A|Out], merge(lx, ly, Out). --(5)
merge(lx, [A|ly], 0) :- true | 0 = [A|Out], merge(lx, ly, Out). --(6)
merge(lx, [], 0) :- true | lx = 0. -----(7)
merge([], ly, 0) :- true | ly = 0. -----(8)

plus1([A|In], 0) :- true | A1 = succ(A), 0 = [A1]. -----(9)

```

Consider the following goal 'top([0], Out)' where 0 is an atom and Out is a variable term. During the execution of this goal, the goal such as merge([0, 0], Oy, Oz) is invoked where Oy and Oz are variable terms. For this goal, the head part of (8) does not match the goal, and (6) and (7) continue to suspend. Thus, only commitment to (5) can make the execution proceed. Thus this program is controlled by the guard part. Continuing the execution, only Out = [0, 0] is derived from top in spite of Out = [0, s(0)] is also an answer in naive declarative sense.

2.2 Goal Forms and \downarrow Annotation

In the axiomatic approach to give semantics for conventional programming languages, the partial correctness of a program is represented in a formula like the following.

input condition {program} output condition

The partial correctness of GHC programs is represented in a similar way. Input conditions and output conditions are predicates over the Herbrand universe constructed from FUN. The semantics of these predicates are relations over the Herbrand universe. We can use for example existentially quantified variables and negations to define the predicates. Thus predicates for input/output conditions can be defined in more natural and intelligible way than the definitions in GHC programs.

A expression representing a set of goal clauses appears in the 'program' part.

Def. 1 : goal form

Let D be a set of guarded clauses. The expression $g : p(t_1, \dots, t_n)$ is said to be a goal form where p is a n-ary predicate name which is defined in D, t_1, \dots, t_n are terms which are defined from FUN and VAR, and Var. 'Var' is a set of meta variables over TERMS and $\text{Var} \cap \text{VAR} = \emptyset$.

In this paper, 'variable' means abstract variables appearing in goal forms, which are denoted by lower case letters x, y, z, u, ... which are not in VAR. Variable terms appearing during the execution of a program which are in VAR (and in TERM) are

denoted by upper case letters U, V, ... Elements of VAR appearing in clauses in D are considered as variables for convenience. In this paper, 'term' means an element of TERM. A term containing an element of Var is called a 'term form'. The set of term forms constructed from Var, VAR and FUN is denoted as 'Term'.

For a goal form g , an individual goal G is derived by applying a substitution $\Sigma : \text{Var} \rightarrow \text{TERM}$. A goal form g can be considered to represent a set of goals $|g|$ as follows.

$$|g| = \{G | \exists \Sigma : \text{Var} \rightarrow \text{TERM}, G = \Sigma g\}$$

For example for $g : \text{merge}(x, [y, y], z)$ set of goals $|g|$ is $\{\text{merge}(X, [Y, Y], Z), \text{merge}(1, [0, 0], W), \text{merge}(X, [0, 0], [0, 1, 0]), \text{merge}([1], [0, 0], []), \dots\}$ Note that an unsuccessful goal is included.

A sequence of goal forms is called a goal clause form. A goal clause form represents a set of goal clauses.

Partial correctness of a GHC program is represented by a formula which contains a top level goal clause between { and }.

Def. 2 : process form

Let D be a set of guarded clauses, g_1, \dots, g_n be a top level goal clause form, then:

- i) g_1, \dots, g_n and each g_i ($1 \leq i \leq n$) are process forms.
- ii) If g is a process form and for some clause in D: $H :- A_1, \dots, A_m | B_1, \dots, B_k$, g can unify with H by the mgu θ , then $\theta B_1, \dots, \theta B_k$ and each θB_h ($1 \leq h \leq k$) are process forms.
- iii) If g is a process form and for some g' and $\Sigma : \text{Var} \rightarrow \text{Term}$ $\Sigma g' = g$, then g' is a process form.

In this paper, it is assumed that for any goal form at most one variable is instantiated by the goal itself. The variable is called the output variable. It is enough to consider that a goal form g represents a set of goals such that $G = \Sigma g$ and Σ substitutes only a variable term to the output variable. Consider a top level goal clause form g_1, \dots, g_n . For a variable x (not output) in g_i which is instantiated by an another process g_j ($i \neq j$) during its execution, the process g_j in which x appears as an output variable is called the producer of x . In this paper, for every non-output variable, its producer is fixed and is not changed by Σ . For every clause in D: $H :- A_1, \dots, A_k | B_1, \dots, B_h$ and for any variable x which appears in B_1, \dots, B_h and does not appear in H, A_1, \dots, A_k , the producer of x is fixed in B_1, \dots, B_h . For a goal form, g with output variable y if g is unifiable with H by mgu θ without instantiating y , then y is also said to be an output variable of a goal clause form, $\theta B_1, \dots, \theta B_h$. These restrictions mean that data-dependencies in programs must be decided obviously.

In the rest of this section \downarrow annotation is introduced. Consider the following example. For a goal form g , a set of all goal clauses which are derived as sequences of subgoals for some instance of g is not represented by a goal clause form which is

derived by symbolic derivation of g on D in general. For example, in Brock-Ackermann's anomaly, the subgoals of goals of the form $\text{top}(x, \text{out})$ have the following form:

$$s(x, \text{mid}, \text{out}), \text{plus1}(\text{out}, \text{mid}). \text{-----} (*)$$

mid is never instantiated by the unification of the goal form and the head part since it is instantiated during execution, and a goal with non-variable term does not appear in execution of any goal in $|\text{top}(x, \text{out})|$, in spite of the form of $(*)$.

It is enough to consider that not only out but also mid is uninstantiated. In this paper, a set of goal clauses which contains variables such as mid is represented by a goal clause form with \downarrow annotation to such variables. The set of subgoals of top is denoted as follows using \downarrow .

$$s(x, \text{mid}\downarrow, \text{out}), \text{plus1}(\text{out}, \text{mid}\downarrow)$$

Namely, for a goal (clause) form g which contains \downarrow annotated variables, $|g|$ is defined as follows.

$$|g| = \{ \Sigma g \mid \exists \Sigma : \text{Var} \rightarrow \text{TERM} \text{ such that for any } \downarrow \text{ annotated variable } x, \Sigma x \in \text{VAR} \}$$

For a goal clause form g_1, \dots, g_n which contains \downarrow annotated variables, $|g_1, \dots, g_n|$ is defined similarly.

In this paper, it is assumed that every goal form contains at most one \downarrow annotated non output variable for simplicity.

For a top level goal clause form g_1, \dots, g_n , if \downarrow annotated variables are contained in g_1, \dots, g_n then the same variables in the process form defined from g_1, \dots, g_n can be \downarrow annotated. Furthermore if y is a variable which appears in $\theta B_1, \dots, \theta B_k$ and appears in neither θH nor $\theta A_1, \dots, \theta A_m$, then y can be annotated, where $H :- A_1, \dots, A_m \mid B_1, \dots, B_k$ is a clause in D such that a process form defined from g_1, \dots, g_n and H can unify with a process form g by the mgu θ .

Def. 3 : Hoare's formula for GHC programs

For a set of guarded clauses D , top level goal clause g_1, \dots, g_n and assertion language L for input/output conditions,

- 1) if $\Phi, \Psi \in L$ then $\Phi \{g_1, \dots, g_n\} \circ \Psi$ is a top level formula.
- 2) if g_1', \dots, g_n' is a process form defined by g_1, \dots, g_n , then $\Phi \{g_1', \dots, g_n'\} \circ \Psi$ is a formula.

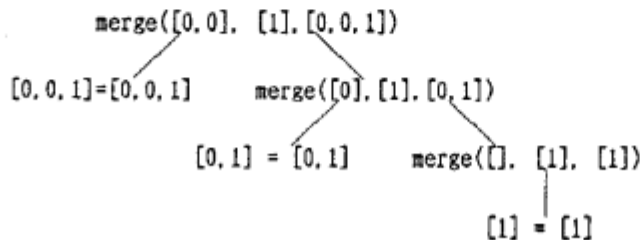
D after $\}$ is abbreviated if there is no confusion. The semantics of the above formula is defined in the following section.

2.3 Operational Semantics of GHC

This section presents an outline of the operational semantics of GHC. The semantics presented here is based on "tree of computation" [Takeuchi 86]. In this paper, the purpose of introducing the notion of the computation tree is to define the semantics of formulas which appear in the proof of partial correctness, so only successful computations are discussed. The semantics of a GHC program is defined as a set of successful computation trees determined from the set of guarded clauses and a goal clause form.

The computation tree for individual goal is defined as the trace tree [Takeuchi 86]. Intuitively, each computation of the GHC program is a tuple of finite trees whose roots are goals. A computation tree is an AND tree formed by a computation. Each node is a goal instantiated by a substitution derived when the computation succeeds. Each child of an internal node is a subgoal of its parent node which is derived when the parent commits to some clause.

The following is an example of computation tree for a goal, $\text{merge}([0,0], [1], Z)$.



Since a GHC program may contain some nondeterminism in general, there are a number of computation trees for a goal and a set of clauses. For a goal clause which consists of several goals executed in parallel, a set of tuples of computation trees $\langle t_1, \dots, t_n \rangle$ is defined similarly. The set of computations defined from a set of guarded clauses D and a goal clause G_1, \dots, G_n is denoted as $\text{COMP}(G_1, \dots, G_n, D)$.

Def. 4 :

For a top level goal clause form g_1, \dots, g_n , the set of computation trees $\text{Comp}(g_1, \dots, g_n, D)$ is defined as follows:

$$\begin{aligned}
 \text{Comp}(g_1, \dots, g_n, D) = \{ \langle t_1, \dots, t_n \rangle \mid & G_1, \dots, G_n \in |g_1, \dots, g_n|, \\
 & \langle t_1, \dots, t_n \rangle \in \text{COMP}(G_1, \dots, G_n, D) \} .
 \end{aligned}$$

It is a little more complicated in the case of the non top level goal form.

In the example in Section 2, 'merge' is invoked with a variable term $0y$ in the second argument, and cannot commit to any clause except (5). Therefore, the goal commits to clause (5) and instantiates its third argument in the form of $[x|y]$. After the producer of the $0y$ receives $[x|y]$, the it is instantiated. In this case, \downarrow means that it does not need to consider the computation that contains commits which require an instantiated term in this variable before the output instantiation which makes the producer active as a computation of this goal form. Thus, the set of computation trees of a non top level process form such as 'merge' is determined by giving D and a set of terms which is substituted for an output variable and activates the producer of the \downarrow

annotated variable. Such set of terms can be represented using the terms which appear in the guards of clauses which define the producer predicate.

In this paper, it is assumed that the set of such terms are represented in a unique term form for simplicity. In other words, the semantics of processes is given as a function from a term form τ to a set of computation trees $\text{Comp}[g_1, \dots, g_n, D](\tau)$.

Def. 5

$\text{Comp}[g_1, \dots, g_n, D](\tau) =$
 $\{ t \mid t \in \text{COMP}(\Sigma g_1, \dots, \Sigma g_n, D), \text{ and the output variable of } g_1, \dots, g_n$
 $\text{can be instantiated more than } \tau \text{ by composing all unifications which}$
 $\text{appear in } t \text{ except subtrees whose root is a goal which makes a non-}$
 $\text{trivial commit about the term form substituted in the } \downarrow \text{ annotated}$
 $\text{variable.} \}$

where a commitment of goal $p(t)$ to a clause C is said to be non-trivial about t if a goal p' which is derived by replacing t by a variable term cannot commit to C . When g_1, \dots, g_n is a top level goal form:

$$\text{Comp}[g_1, \dots, g_n, D](t) = \text{Comp}(g_1, \dots, g_n, D)$$

where t is a term form which represents a set of terms such that g_1, \dots, g_n cannot output.

Def. 6 :

Let g_1, \dots, g_n be a non top level goal clause form and Γ be a set of formulas which are the form $\Theta \{g\} T$ where g is one of the process forms which are defined from g_1, \dots, g_n . For g_1, \dots, g_n , a set of hypotheses Γ and a term form τ :

$$\models \Phi \{g_1, \dots, g_n\} \Psi$$

iff

for all $\langle t_1, \dots, t_n \rangle \in \text{Comp}[g_1, \dots, g_n, D](\tau)$ such that $\langle t_1, \dots, t_n \rangle \in \text{COMP}(\Sigma g_1, \dots, \Sigma g_n, D)$ and the root of each t_i ($1 \leq i \leq n$) is $\sigma \Sigma g_1, \dots, \sigma \Sigma g_n$, if all of Γ is true as top level then $\Sigma \Phi \Rightarrow \sigma \Sigma \Psi$. A formula, $\Theta \{g\} T$ is said to be true as top level when for all $t \in \text{Comp}(g, D)$ if the root of t is $\sigma \Sigma g$ then $\Sigma \Theta \Rightarrow \sigma \Sigma T$.

Def. 7 :

A top level goal clause form g_1, \dots, g_n is partially correct wrt Φ and Ψ iff Γ is an empty set and

$$\models \Phi \{g_1, \dots, g_n\} \Psi.$$

In other words, a top level goal clause form g_1, \dots, g_n is partially correct wrt Φ and Ψ if and only if for all $\langle t_1, \dots, t_n \rangle \in \text{Comp}(g_1, \dots, g_n, D)$ if $\langle t_1, \dots, t_n \rangle \in \text{COMP}(\Sigma g_1, \dots, \Sigma g_n, D)$ and the root of each t_i ($1 \leq i \leq n$) is $\sigma \Sigma g_1, \dots, \sigma \Sigma g_n$ then $\Sigma \Phi \Rightarrow \sigma \Sigma \Psi$.

3. Axiom System

The axiom system presented here is based on the following idea. The property of a goal clause form g_1, \dots, g_n is derived from the property of each g_i ($1 \leq i \leq n$). The property of each g_i is derived from the properties of subgoals. An induction method is adopted for the proof of recursive predicates.

Inference rules

$$\text{Substitution: } \frac{\Phi \{g\} \Psi}{\sigma \Phi \{\sigma g\} \sigma \Psi}$$

where σ does not instantiate any variable annotated with \downarrow .

$$\text{Consequence 1 } \frac{\Phi \{g_1, \dots, g_n\} \Psi \quad \Psi \Rightarrow \Psi'}{\Phi \{g_1, \dots, g_n\} \Psi'}$$

$$\text{Consequence 2 } \frac{\Phi' \Rightarrow \Phi \quad \Phi \{g_1, \dots, g_n\} \Psi}{\Phi' \{g_1, \dots, g_n\} \Psi}$$

$$\text{Derivation 1 } \frac{\Phi \wedge T=S \Rightarrow \Psi}{\Phi \{T=S\} \Psi}$$

$$\text{Derivation 2 } \frac{P_1, \dots, P_s}{\Phi \{g\} \Psi}$$

where P_1, \dots, P_s is the sequence of all P_j ($1 \leq j \leq s$) defined as follows.

There is a guarded clause : $H_j :- B_{j1}, \dots, B_{jh_j} \mid A_{j1}, \dots, A_{jm_j}$,

in D for j ($1 \leq j \leq s$) such that H_j is unifiable with g ($\sigma, g =$

σ, H_j), σ , does not instantiate the variable annotated with

\downarrow in the unification of a term form appearing in g , and P_j has the following form:

$$P_j = \bigwedge_{k=1, h_j} \sigma, B_{jk} \wedge \sigma, \Phi \{\sigma, A_{j1}, \dots, \sigma, A_{jm_j}\} \sigma, \Psi$$

where for each B_{jk} ($k=1, h_j$), there is a substitution λ_{jk} such that $\lambda_{jk} B_{jk}$ is true and does not instantiate any variable in B_{jk} annotated with \downarrow .

The inference using this rule with variables with \downarrow in its conclusion is called degenerated inference when the formula which was obtained by deleting \downarrow from the conclusion cannot be inferred from formulas which were obtained by deleting all \downarrow from premises of this inference.

The rule, Parallel is introduced. This rule takes formulas for the properties of each

process g_i ($1 \leq i \leq n$) as premises and takes a formula for the properties of g_1, \dots, g_n as a conclusion. An inference using this rule is valid only if a certain condition is satisfied on the sub proof schema P whose root is the result of application of the Parallel rule. The notion of a sub proof schema is defined as a subtree of a proof schema defined below. Two propositions $R(x, \tau, \text{form}(g_1), fr, P)$ and $O(x, \tau, \text{form}(g_2), fr, P)$ are defined where x is a variable, τ is an element of Term, g_1 is a goal form which contains x as a non output variable, g_2 is a goal form which contains x as an output variable, $\text{form}(g)$ is a formula which appears in P and takes the form $\Phi \{g\} \Psi$, and fr is a conclusion of degenerated inference.

$R(x, \tau, \text{form}(g_1), fr, P) =$
 if $\lceil \text{if } \text{form}(g_1) \text{ is the form of } \Theta \{g_1\} T$
 then $x = \tau \wedge \Theta$ is equal to false. \rceil then true
 else if \lceil there appears a producer p of x in P \rceil
 then $O(x, \tau, \text{form}(p), fr, P)$
 else true

where p is the producer of x . Intuitively, $R(x, \tau, \text{form}(g_1), fr, P)$ means that x cannot take the form of τ when g_1 is invoked.

$O(x, \tau, \text{form}(g_2), fr, P) =$
 if $\lceil fr \text{ is } \text{form}(g_2) \rceil$ then true
 else if $\lceil g_2$ contains a unification of x and a term form $t \rceil$ then
 if $\lceil t$ and τ are unifiable \rceil then if $\lceil \exists \sigma : t = \sigma \tau \rceil$ then false
 else $\lceil \bigvee_{i=1, h} O(x_i, \sigma x_i, \text{form}(p_i), fr, P) \rceil$
 where σ is a substitution such that $\sigma t = \sigma \tau$ and
 instantiates variables, x_1, \dots, x_h appears in t , and
 p_i is the producer of x_i . \rceil
 else true
 else
 $\lceil \bigwedge_{1 \leq k \leq m} \bigvee_{1 \leq u \leq w} R(y_u, \sigma_k y_u, \text{form}(g_2), fr, P) \vee$
 $O(x, \tau, \text{form}(q_k(\dots, x)), fr, P) \rceil$
 where there exists a clause $: H_k(\dots, y) :- B_k \mid \dots, q_k(\dots, y), \dots$
 ($1 \leq k \leq m$) such that for some substitution $\sigma_k : \sigma_k g_2 = \sigma_k H_k$, $\sigma_k y$
 = x and σ_k instantiates variables y_1, \dots, y_w appears in g_2 . \rceil

$O(x, \tau, \text{form}(g_2), \text{form}(gr), P)$ means that g_2 cannot make x the form of τ without executing gr .

Parallel:

For a set of goal forms $\{g_1, \dots, g_n\}$, if g_i contains a variable x with the \downarrow annotation and there exists a producer of x , g_j ($1 \leq j \leq n$), then let g_i' be a goal form deleting \downarrow from x otherwise $g_i' = g_i$.

If for all degenerated inference contained in the sub proof schema of $\Phi_i \{g_i\} \Psi_i$ ($1 \leq i \leq n$):

$$\bigwedge_{1 \leq j \leq m} R(x_j, \tau_j, \Theta_j \{h_j\} T_j, \Theta_j \{h_j\} T_j, P) = \text{true}$$

then:

$$\frac{\Phi_1 \{g_1\} \Psi_1, \dots, \Phi_n \{g_n\} \Psi_n}{\bigwedge_{i=1, n} \Phi_i \{g_i', \dots, g_n'\} \bigwedge_{i=1, n} \Psi_i}$$

where $\Theta_j \{h_j\} T_j$ ($1 \leq j \leq m$; m is the number of degenerated inference) is the conclusion of each degenerated inference, x_j is the variable which makes the inference degenerated, and τ_j is a term form which failed to unify with x_j because of \downarrow .

For a non top level process form $p(x \downarrow, \dots)$, when a sub proof schema for $\Phi \{p(x \downarrow, \dots)\} \Psi$ with degenerated inference for x is constructed, it means that if x is instantiated with some time delay then the result of the computation satisfies Φ for all input which satisfies Ψ under some assumptions. Furthermore, if the Parallel inference rule can be applied to the sub proof schema of $\Phi \{p(x \downarrow, \dots)\} \Psi$ and the sub proof schema of the producer of x , then it means that the time delays where the producer outputs x and which are considered for the sub proof schema of $\Phi \{p(x \downarrow, \dots)\} \Psi$ are consistent.

Read:

$$\frac{\Phi \{g(\dots, x, \dots)\} \Psi}{\Phi \{g(\dots, x \downarrow, \dots)\} \Psi}$$

where \downarrow is attached to all occurrences of x in $g(\dots, x \downarrow, \dots)$.

In this system, all formulas which are true in the domain of the program are regarded as an axiom like the usual Hoare-like system. In addition, the followings are introduced as the axioms.

Axiom

- 1) false $\{g_1, \dots, g_n\} \Psi$
- 2) $\Phi \{g_1, \dots, g_n\}$ true

In most Hoare-like proof systems, a proof schema is defined as a tree in which each of the leaves corresponds to an axiom and the root corresponds to the formula which expresses partial correctness. In this system, in addition to axioms, 'the hypothesis of induction' can appear as a leaf.

Def. 8:

For a top level goal clause form g_1, \dots, g_n , a proof schema of formula $\Phi \{g_1, \dots, g_n\} \Psi$ is a tree such that:

- 1) The root of the tree corresponds to $\Phi \{g_1, \dots, g_n\} \Psi$.
- 2) For every node n , either a) or b) following is true.
 - a) For some inference rule (shown in Section 3), n is an instance of a conclusion and each child of n corresponds to a premise.
 - b) n is a leaf and one of the following is true:
 - (i) n is an axiom.
 - (ii) n is identical to one of its ancestors n' , the Derivation 2) rule is used at least once on the path from n' to n and n does not contain the \downarrow annotated variable as non output variable.

For a goal clause form g_1, \dots, g_n , if there exists a sub proof schema of $\Phi \{g_1, \dots, g_n\} \Psi$ for some Φ and Ψ with formulas f_1, f_2, \dots, f_k which are not axioms appearing as the leaves then :

$$\models \Phi \{g_1, \dots, g_n\} \Psi$$

for $\Gamma = \{f_1, f_2, \dots, f_k\}$ and τ , where τ is the result of compositions of all unifications for the output variable which appear in the sub proof schema and are not children of any degenerated inference.

Especially for a top level goal clause form g_1, \dots, g_n , if there exists a proof schema of $\Phi \{g_1, \dots, g_n\} \Psi$ for some Φ and Ψ , then g_1, \dots, g_n is partially correct wrt Φ and Ψ .

Using this axiom system, the following property of Brock Ackermann anomaly can be proved.

$$[a] = \text{in } \{\text{top}(\text{in}, \text{out})\} \text{ out} = [a, a]$$

This property of top is made true by the guard/commit mechanism of GHC, and cannot prove with a verification method for pure Horn logic programs.

5. Conclusion

This paper proposed an axiom system for proving the partial correctness of GHC programs. In this system, the partial correctness of programs which are executed deterministically by the guard/commit mechanism can be proved for enough strong output conditions.

In this paper, a number of restrictions to GHC programs were assumed. However no method that decides if a program satisfies the restriction condition or not is presented here for the restrictions about obvious data-dependency. Namely we expect some dynamic analysis method for deciding if the output variable of a program is fixed uniquely. However such dynamic analysis method for GHC programs is not investigated enough yet. It is one of the important topic for future research. We consider that verification method of programs such that presented here are useful for the foundations of investigation of analysis method of GHC programs.

Acknowledgment

I would like to thank Dr. K. Furukawa, and all the members of the First Laboratory of ICOT for many useful discussions.

References:

- [Brock 81] J. D. Brock, W. B. Ackermann, Scenarios: A Model of Non- determinate Computation, Lecture Notes in Computer Science, No. 107 Springer, 1981
- [Clark 86] K. L. Clark and S. Gregory, PARLOG: Parallel programming in logic, ACM Trans. on Programming Language and Systems 86, 1986
- [Kameyama 87] Y. Kameyama, Axiomatic System for Concurrent Logic Programming Languages, Master's Thesis of the University of Tokyo, 1987
- [Kanamori 86] T. Kanamori and H. Seki, Verification of Prolog Programs Using an Extension of Execution, Lecture Notes in Comp. Sci., No. 225, 1986
- [Levi 87] G. Levi and C. Palamidessi, An Approach to the Declarative Semantics of Synchronization in Logic Language, Proc. of International Conf. on Logic Programming 87, 1987
- [Maher 87] M. J. Maher, Logic Semantics for a Class of Committed-Choice Programs, Proc. of International Conf. on Logic Programming 87, 1987
- [Murakami 87] M. Murakami, Proving Partial Correctness of Guarded Horn Clauses, The Logic Programming Conference '87 1987
- [Saraswat 85] V. A. Saraswat, Partial Correctness Semantics for CP [\downarrow , $\&$], Lecture Notes in Comp. Sci., No. 206, 1985
- [Saraswat 87] V. A. Saraswat, The Concurrent logic programming CP: definition and operational semantics, Proc. of ACM Symp. on Principles of Programming Languages, 1987
- [Shapiro 86] E. Y. Shapiro, Concurrent Prolog: A progress report, Lecture Notes in Comp. Sci. No. 232, 1986
- [Takeuchi 86] A. Takeuchi, Towards a Semantic Model of GHC, Tech. Rep. of IECE, COMP86-59, 1986
- [Ueda 85] K. Ueda, Guarded Horn Clauses, Tec. Rep. of ICOT, TR-103, 1985
- [Ueda 86] K. Ueda, On Operational Semantics of Guarded Horn Clauses, Tech. Memo of ICOT, TM-0160, 1986