ソフトウェア基礎論24—4
(1988. 2. 12)

An Axiomatic Verification Method for Synchronizations of GHC 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 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 occuers as an input variable or an output, and for any variable which is shared between more than two processes, which process instanciats the variable can be decided uniquely. These conditions are assumed because that the dependencies are refered 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 :- B1,…, Bn | A1, …, Am.

where H is the head of the clause, H, B1,…,Bn is the guard, and A1, …, Am is the body. Note that the clause head is included in the guard. Each Bi($1 \leqq i \leqq n$) has the form 'true' or T = S, where T and S are in the set of terms, TERM, constructed from FUN and VAR. Each Aj($1 \leqq j \leqq m$) takes the form p(T1,…Tk) or T = S, where p $\in$ PRED and Ti($1 \leqq i \leqq k$) $\in$ TERM. H takes the form p(T1,…Tk). The operator '|' is called the commitment operator. A goal clause takes the form of a body part and is denoted:

G1, …, Gh

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

Dba:

```
top(In, Out) :- true |
      s(In, Mid, Out), plus1(Out, Mid). --------(1)

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

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

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

merge([A|Ix], Iy, O) :- true |
             O = [A|Out], merge(Ix, Iy, Out). --(5)
merge(Ix, [A|Iy], O) :- true |
             O = [A|Out], merge(Ix, Iy, Out). --(6)
merge(Ix, [], O) :- true | Ix = O.  ------------(7)
merge([], Iy, O) :- true | Iy = O. -------------(8)

plus1([A|In], O) :- true |
          A1 = succ(A), O = [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 ↓ 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, …, t n) is said to be a <u>goal form</u> where p is a n-ary predicate name which is defined in D, t 1, …, t n are terms which are defined from FUN and VAR, and Var. 'Var' is a set of meta variables over TERMS and Var ∩ VAR = ϕ.

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 Σ : Var → TERM. A goal form g can be considered to represent a set of goals | g | as follows.

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

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

A sequence of goal forms is called a <u>goal clause form</u>. 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, …, g n be a top level goal clause form, then:

i) g 1, …, g n and each g i (1≦i≦n) are process forms.
ii) If g is a process form and for some clause in D: H :- A1, …, Am | B 1, …, B k, g can unify with H by the mgu θ, then θ B 1, …, θ B k and each θ B h (1≦h≦k) are process forms.
iii) If g is a process form and for some g' and Σ :Var → Term Σ 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 = Σ g and Σ substitutes only a variable term to the output variable. Consider a top level goal clause form g 1, …, g n. For a variable x (not output) in g i which is instantiated by an another process g j (i ≠ 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, …, A k | B 1, …, B h and for any variable x which appears in B 1, …, B h and does not appear in H, A 1, …, A k, the producer of x is fixed in B 1, …, 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, θ B 1, …, θ B h. These restrictions mean that data-dependencies in programs must be decided obviously.

In the rest of this section ↓ annotation is introduced. Consider the following example. For a goal form g, a set of all goal clauses which are deirved 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 top(x, o ut) have the following form:

s(x, mid, o ut), plus1(o ut, mid). -------(*)

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 |top(x, o ut)|, in spite of the form of (*).

It is enough to consider that not only o ut 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 ↓ annotation to such variables. The set of subgoals of top is denoted as follows using ↓.

s(x, mid↓, o ut), plus1(o ut, mid↓)

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

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

For a goal clause form g1, …, gn which contains ↓ annotated variables, |g1,…, gn| is defined similarly.

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

For a top level goal clause from g1, …, gn, if ↓ annotated variables are contained in g1, …, gn then the same variables in the process form defined from g1, …, gn can be ↓ annotated. Furthermore if y is a variable which appears in θB1, …, θBk and appears in neither θH nor θA1, …, θAm, then y can be annotated, where

H :- A1, …, Am | B1, …, Bk is a clause in D such that a process form defined from g1, …, gn 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 g1, …, gn and assertion language L for input/output conditions,

1) if Φ, $\Psi \in$ L then Φ {g1, …, gn}$_D$ Ψ is a top level formula.
2) if g1', …, gm' is a process form defined by g1, …, gn, then Φ {g1', …, gm'}$_D$ Ψ is a formula.

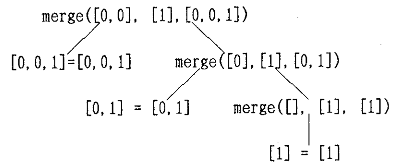
D after } is abbreviated if there is no confusion. The semantics of the above formula is defined in the following section.

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, `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 <t1,…, tn> is defined similarly. The set of computations defined from the set of guarded clauses D and goal clause G1,…,Gn is denoted as COMP(G1,…,Gn, D).

Def. 4 :

For a top level goal clause form g1, …, gn, the set of computation trees Comp(g1, …, gn, D) is defined as follows:

Comp(g1, …, gn, D) = {<t1,…, tn> | G1, …, Gn $\in$ |g1, …, gn|, <t1,…, tn> $\in$ COMP(G1, …, Gn, D)} .

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 Oy 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 Oy receives [x|y], the it is instantiated. In this case, ↓ 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 ↓ 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 Comp[g1,…, gn, D](τ).

Def. 5

Comp[g1,…, gn, D](τ) =

{t | t ∈ COMP(Σg1,…, Σgn, D), and the output variable of g1,…, gn can be instantiated more than τ by composing all unifications which appear in t except subtrees whose root is a goal which makes a non- trivial commit about the term form substituted in the ↓ annotated 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 g1, …, gn is a top level goal form:

Comp[g1,…, gn, D](t) = Comp(g1, …, gn, D)

where t is a term form which represents a set of terms such that g1, …, gn cannot output.

Def. 6 :

Let g1,…, gn be a non top level goal clause form and Γ be a set of formulas which are the form Θ {g} Υ where g is one of the process forms which are defined from g1,…, gn. For g1,…, gn, a set of hypotheses Γ and a term form τ :

$$\models \Phi \ \{g1,\cdots, gn\} \ \Psi$$
$$\text{iff}$$

for all <t1, …, tn> ∈ Comp[g1,…, gn, D](τ) such that <t1, …, tn> ∈ COMP(Σg1, …, Σgn, D) and the root of each ti (1≦i≦n) is σΣg1,…, σΣgn, if all of Γ is true as top level then ΣΦ => σΣΨ. A formula, Θ {g} Υ is said to be <u>true as top level</u> when for all t ∈ Comp(g, D) if the root of t is σΣg then ΣΘ ⇒ σΣΥ.

Def. 7 :

A top level goal clause form g1,…, gn is partially correct wrt Φ and Ψ iff Γ is an empty set and

$$\models \Phi \ \{g1,\cdots, gn\} \ \Psi.$$

In other words, a top level goal clause form g1,…, gn is partially correct wrt Φ and Ψ if and only of for all <t1, …, tn> ∈ Comp(g1,…, gn, D) if <t1, …, tn> ∈ COMP(Σg1,…, Σgn, D) and the root of each ti (1≦i≦n) is σΣg1,…, σΣgn then ΣΦ => σΣΨ.

3. Axiom System

The axiom system presented here is based on the following idea. The property of a goal clause form g1, …, gn is derived from the property of each gi (1≦i≦n). The property of each gi is derived from the properties of subgoals. An induction method is adopted for the proof of recursive predicates.

Inference rules

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

where σ does not instantiate any variable annotated with ↓.

Consequence 1
$$\frac{\Phi \ \{g1,\ \cdots,\ gn\} \ \Psi \qquad \Psi \Rightarrow \Psi'}{\Phi \ \{g1,\ \cdots,\ gn\} \ \Psi'}$$

Consequence 2
$$\frac{\Phi' \Rightarrow \Phi \qquad \Phi \ \{g1,\ \cdots,\ gn\} \ \Psi}{\Phi' \ \{g1,\ \cdots,\ gn\} \ \Psi}$$

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

Derivation 2
$$\frac{P1,\ \cdots,\ Ps}{\Phi \ \{g\} \ \Psi}$$

where P1, …, Ps is the sequence of all Pj (1 ≦ j ≦ s) defined as follows. There is a guarded clause : Hj :- Bj1, …, Bjh_j | Aj1, …, Ajm_j in D for j (1 ≦ j ≦ s) such that Hj is unifiable with g ($\sigma_j g = \sigma_j Hj$), σ_j does not instantiate the variable annotated with ↓ in the unification of a term form appearing in g, and Pj has the following form:

$$P_j \equiv \bigwedge_{k=1,\ h_j} \sigma_j B_{jk} \wedge \sigma_j \Phi$$
$$\{\sigma_j A_{j1}, \cdots, \sigma_j A_{jm}\}\ \sigma_j \Psi$$

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 ↓.

The inference using this rule with variables with ↓ in its conclusion is called <u>degenerated inference</u> when the formula which was by derived deleting ↓ from the conclusion cannot be inferred from formulas which were derived by deleting all ↓ from premises of this inference.

The rule, Parallel is introduced. This rule takes formulas for the properties of each process g_i ($1 \leqq i \leqq n$) as a premise and takes a formula for the properties of $g_1, \cdots, g_n$ 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 $R(x, \tau, form(g_1), fr, P)$ and $O(x, \tau, form(g_2), fr, P)$ 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, 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, τ, form(g1), fr, P) =
  if 「if form(g1) is the form of Θ {g1} T
  then x = τ ∧ Θ is equal to false.」 then true
  else if 「there appears a producer p of x in P」
    then O(x, τ, form(p), fr, P)
    else true

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

O(x, τ, form(g2), fr, P) =
  if 「fr is form(g2)」 then true
  else if 「g2 contains a unification of x and a
      term form t」 then
  if 「t and τ are unifiable」 then
    if 「∃ σ : t = στ」 then false
    else 「$\bigvee_{i=1,\ h}$ O(xi, σxi, form(pi), fr, P)
      where σ is a substitution such
      that σt = στ and instantiates
      variables, $x_1, \cdots, x_h$ appears in
      t, and pi is the producer of
      xi.」
  else true
else
  「$\bigwedge_{1 \leqq k \leqq m} (\bigvee_{1 \leqq u \leqq w}$ R(yu, σkyu, form(g2), fr, P) ∨
    O(x, τ, form(qk(…, x)), fr, P))
  where there exists a clause : Hk(…, y)
  :- Bk | …, qk(…, y), …. (1 ≦ k ≦ m)
  such that for some substitution σk :
  σkg2 = σkHk, σy = x and σk
  instantiates variables $y_1, \cdots, y_w$
  appears in g2.」

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

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

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

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

then:

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

where $\Theta_j\ \{h_j\}\ T_j$ ($1 \leqq j \leqq m$: 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 ↓.

For a non top level process form p(x↓, …), when a sub proof schema for Φ {p ($x\downarrow$,…)} Ψ 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 Φ {p($x\downarrow$,…)} Ψ 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 Φ {p($x\downarrow$, …)} Ψ are consistent.

Read:

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

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

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 {g1, …, gn} Ψ

2) Φ {g1, …, gn} 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 g1 ,…, gn, a proof schema of formula Φ {g1, …, gn } Ψ is a tree such that:

1) The root of the tree corresponds to Φ {g1, …, gn } Ψ .

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 form n' to n and n does not contain the $\downarrow$ annotated variable as non output variable.

For a goal clause form g1,…, gn, if there exists a sub proof schema of Φ {g1,…, gn} Ψ for some Φ and Ψ with formulas f1, f2,… , fk which are not axioms appearing as the leaves then :

$$\models\ \Phi\ \{g1,\cdots, gn\}\ \Psi$$

for Γ = {f1, f2,… , fk} 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 g1,…, gn, if there exists a proof schema of Φ {g1,…, gn} Ψ for some Φ and Ψ, then g1,…, gn is partially correct wrt Φ and Ψ.

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

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

The outline of the proof is presented in the appendix. This property of top is made true by the guard/commit mechanism of GHC, and cannot be discussed by regarding the programs as predicates. Consider the program obtained by replacing the clause (3) by the following two clauses.

```
pop([A|B], O) :- true | O = [A, O1], pop1(B, O1).
pop1([A|_], O) :- true | O = A.
```

Although the new program is equivalent to top in the declarative sense, it does not satisfy this property. Of course, this property cannot be proved for the new program by this axiom system. Namely in the proof of the appendix, if the sub proof schema of (a,16) is replaced by the proof schema for new definition of pop in the inference which derives (a,17) by the Parallel rule, then R(oy, [A|Iy],(a,6), (a,6), P17') is not true where P17' is the new proof schema. Thus, the Parallel rule cannot be applied and the proof of this property fails. Thus the proof fails. Of course it is easy to show that it is immposible to prove (a,22) with another rule in this system.

However, the following property is true for the new program and it can be proved by this system.

$$[a] = in \ \{t2(in, out)\} \ out = [a, a] \vee out = [a, s(a)]$$

4. 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. A method that decides if a program satisfies the restriction or not is not presented here for the restrictions about obvious data-dependency. Namely we expect some dynamic anlysis 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. The investigation of analysis method 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 Commited-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 [↓, |, &], 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

Appendix : Example

Verification of the Brock-Ackermann anomaly

The outline of the proof X= [a] {top(X,Y)} X=[a,a] is presented.

It is easy to show from the axiom and Derivation 1 :

$$true \ \{oz0 = [a0|oz1]\} \ oz0 = [a0|oz1]. \quad \text{---(a ,1)}$$

The following is the axiom.

$$true \ \{merge(ix0, \ oy\downarrow, \ oz1)\} \ true \quad \text{---------(a ,2)}$$

Clearly, Parallel can be applied to (a,1) and (a,2).

$$true \ \{oz0 = [a0|oz1], merge(ix0, \ oy\downarrow, \ oz1)\} \ oz0 = [a0|oz1] \quad \text{-------------(a,3)}$$

Applying Consequence:

$$true \ \{oz0 = [a0|oz1], merge(ix0, \ oy\downarrow, \ oz1)\} \ [a1|ix0] = [a0] \Rightarrow oz0 = [a0|\_]. \quad \text{----- (a,4)}$$

The following are derived from the axiom and Consequence:

$$true \ \{oy\downarrow = oz0\} \ [] = [a0] \Rightarrow oz0 = [a0|\_] \quad \text{-----------(a,5)}$$

Applying Derivation 2 to (a,4) and (a,5), (a,6) is derived. This is a degenerate inference.

$$true \ \{merge(ix, \ oy\downarrow, \ oz0)\} \ ix = [a0] \Rightarrow oz0 = [a0|\_] \quad \text{----------(a,6)}$$

On the other hand, the following is the axiom and Derivation 1:

$$true \ \{oz = [a2|oz0]\} \ oz0 = [a0|\_] \Rightarrow oz = [a2, \ a0|\_] \quad \text{------ (a,7)}$$

The following are shown from the definition:

R(o y, [A|Iy], (a, 6), (a, 6), P8) = true
R(o y, [], (a, 6), (a, 6), P8) = true

where P8 is the sub proof schema which results in the application of Parallel to (a,6) and (a,7) as the root. Thus, Parallel can be applied to (a, 6) and (a, 7).

true {o z = [a 2| o z0], merge(i x, o y↓, o z0)}
o z0 = [a 0|_] ⇒ o z = [a 2, a 0|_] ∧
i x = [a 0] ⇒ o z0 = [a 0|_] -------- (a,8)

Applying Consequence :

true {o z = [a 2| o z0], merge(i x, o y↓, o z0)}
[a 2| i x] = [a, a] ⇒ 0z = [a, a |_].
-------------(a,9)

The following is from the axiom and Consequence :

true {o y↓ = o z}
[] = [a, a] ⇒ o z = [a, a |_]. -------(a,10)

Applying Derivation 2 to (a,9) and (a,10) , (a,11) is derived. This is a degenerate inference about o y↓.

true {merge(o x, o y↓, o z)}
o x = [a, a] ⇒ o z = [a, a |_] -----(a,11)

The following is the axiom.

true {dup(m↓, o y)} true. ---------(a,12)

The following is from the axiom and Derivation 1 :

[a] = [a 3| i n1] {o x = [a 3, a 3]}
o x = [a, a] .--------------(a,13)

Applying Derivation 2 :

[a] = i n {dup(i n, o x)} o x = [a, a].
-----------------(a,14)

It is easy to show the axiom and Derivation 1:

true {o ut = [a 4, a 4]}
[a 4, a 4|_] = [a, a |_]
⇒ o ut = [a, a]. ------(a,15)

Applying Derivation 2 :

true {pop(o z, o ut)}
o z = [a, a |_]
⇒ o ut = [a, a] ------------(a,16)

The following are shown from the definition:

R(o y, [A|Iy], (a, 6), (a, 6), P17) = true
R(o y, [], (a, 6), (a, 6), P17) = true
R(o y, [A|Iy], (a, 11), (a, 11), P17) = true
R(o y, [], (a, 11), (a, 11), P17) = true

where P17 is the sub proof schema which results in the application of Parallel to (a, 11), (a, 12), (a,14) and (a, 16) as the root. Thus, Parallel can be applied to (a, 11), (a, 12), (a,14) and (a, 16).

[a] = i n
{dup(i n, o x), dup(m↓, o y),
merge(o x, o y, o z), pop(o z, o ut) }
o x = [a, a] ∧ o x = [a, a]
⇒ o z = [a, a |_].
------(a,17)

Applying Consequence:

[a] = i n {dup(i n, o x), dup(m↓, o y),
merge(o x, o y, o z), pop(o z, o ut) }
o ut = [a, a].
-----------(a,18)

Applying Derivation 2 :

[a] = i n {s(i n, m↓, o ut) } o ut = [a, a].
--------------(a,19)

The following is the axiom.

true {plus1(o ut, m↓)} true --------------(a,20)

The following are shown from the definition:

R(o y, [A|Iy], (a, 6), (a, 6), P21) = true
R(o y, [], (a, 6), (a, 6), P21) = true
R(o y, [A|Iy], (a, 11), (a, 11), P21) = true
R(o y, [], (a, 11), (a, 11), P21) = true

where P21 is the sub proof schema which has the result of applying Parallel to (a, 19) and (a, 20) as the root. Thus, Parallel can be applied to (a, 19) and (a, 20).

[A] = i n {s(i n, m↓, o ut),
plus1(o ut, m↓)} o ut = [a, a]
-----------(a,21)

Applying Derivation 2:

[a] = i n {top(i n, o ut)} o ut = [a, a] ---- (a,22)