TR-242

Proving Partial Correctness of Guarded Horn Clauses Programs

by

M. Murakami

March, 1987

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

Proving Partial Correctness of Guarded Horn Clauses Programs

Masaki Murakami

Institute for New Generation Computer Technology,
Tokyo, Japan

ABSTRACT: Guarded Horn Clauses (GHC in short) [Ueda 85] is a parallel programming language based on Horn logic. A verification method of partial correctness for GHC programs is discussed here. The author investigated a Hoare-like axiomatic system for proving partial correctness of GHC programs. This article presents fragments of the axiomatic system. Programs which generate processes dynamically during the execution or which contain control of nondeterminism by the guard mechanism are verified by this systems.

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 communications and synchronization in logic programming framework. In these languages, Horn logic is extended in order to describe these notions. For example, a GHC program consists of a finite set of Horn clauses with a 'guard' part. It is not enough in such languages to regard a program as a set of Horn clauses to give semantics to the program. For example, [Takeuchi 86] introduced two GHC programs which are equivalent in the declarative sense but the results of executions are not identical.

Results of the formal semantics of such languages have been reported in several sources, particularly the semantics of their synchronization mechanisms [Ueda 86, Takeuchi 86, Levi 85, 87, Saraswat 85]. However, since most of them are based on the operational or an extension of the declarative approach. It is too difficult to apply these semantics to prove the properties of given programs.

Several verification methods have been reported for pure Horn logic programming languages, for example, those method which are based on extended execution [Kanamori 86] or attribute grammar [Deransart 87]. However, in most of them, programs which can be proved are 'pure' programs which do not contain any extra logical feature such as the cut ! operator. In GHC, the synchronization of processes is achieved by the guard part. Nondeterminism of a program is controlled by guard mechanisms. Since the result of the execution of programs is affected by guard mechanism control, it does not seem a good way to extend the verification methods for Prolog programs to GHC programs.

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

Several verification systems have been reported for traditional parallel programming languages such as communicating sequential processes (CSP) [Saundararajan 84, Murakami 86], but not have been reported for parallel programming languages based on Horn logic. Verification systems for languages such as GHC needs to have different features from the verification

systems of traditional parallel programming languages. In the systems for traditional programming languages already reported, the number of processes is fixed for any input value before a program executed and processes cannot be generated dynamically during its execution. It is difficult to formalize in an axiomatic way the programming languages which contain dynamic generation of processes. In GHC, processes are presented as goals, so they are generated dynamically as the example of a program to find the maximum element from a given list in the tournament manner. (see Section. 3). A verification system for GHC programs should be powerful enough to prove such program.

The author investigated an axiomatic system for proving partial correctness of GHC programs. This article presents fragments of the axiomatic system.

Section 2 of this article contains a brief introduction of GHC and a definition of partial correctness. Section 3 introduces a fragment of the axiomatic system with a simple example of proof. It is an elementary part of the system. The section shows how the partial correctness of programs which contain dynamic generation of processes are proved using an induction method. Section 4 presents a refinement of the proof system by which programs with control of nondeterminism by guard mechanisms can be verified.

A full set of axiomatic system and discussions on soundness and completeness of the system will appear in [Murakami 87].

2. Partial correctness of GHC

2.1 Guarded Horn Clauses

Guarded Horn Clauses (GHC) [Ueda 85] is a parallel logic programming language. For a set of predicate symbols PRED, function symbols FUN and variable symbols VAR, a program of GHC consists of a finite set of guarded clauses. A guarded clause has the form:

$$H :\sim B1, \cdots, Bn \mid A1, \cdots, Am.$$

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

$$G1, \cdots, Gh$$

where each Gi ($1 \leqq i \leqq h$) is called a goal.

The computation rule of GHC program is as follows. To execute a program is to refute a given goal by means of input resolution using the clauses forming the program. This can be done in a fully parallel manner i.e. each Gi ($1 \leqq i \leqq h$) is solved in parallel and each clause is tested in parallel for each Gi under the following rules.

(a) Solving the guard part:
For a goal G and a clause C : H:- B1,…, Bn | A1, …, Am if G and H can unify without instantiating any variable term (substituting any non-variable term into variable term) in G and B1,…, Bn is solved without instantiating any variable in G then the guard part of C succeeds.

(b) Suspend:
For a goal G and a clause C if B1,…, Bn or unification between G and H cannot succeed without instantiating variable terms in G, then G suspends with C until the variable terms in G are instantiated by some other goal.

(c) Commitment:
For a goal G and a clause C, when the guard part of C succeeds, it is first confirmed that no other clause has been committed for G. If confirmed, C is selected exclusively for subsequent of goal. Unification invoked in the active part of clause C cannot instantiate the goal G until C is commited to G.

A goal which has the form 'true', atom(T), T = S or T > (<) S etc. is solved as a built-in predicate.

Example:

The following is an example of a GHC program that takes two list in first and second argument as inputs, merges them and returns the result to the third argument. Lists are denoted using the syntax as DEC-10 Prolog.

```
merge([A|Ix], Iy, O) :- true | O = [A|Out], merge(Ix, Iy, Out).
merge(Ix, [A|Iy], O) :- true | O = [A|Out], merge(Ix, Iy, Out).
merge(Ix, [], O) :- true | Ix = O.
merge([], Iy, O) :- true | Iy = O.
```

If the goal has the form of:

```
merge([1,2], V, Z)
```

where V and Z are variable terms, then only the first clause can commit with this goal and the second and the third clause suspend. The fourth clause does not matchs the goal. Thus, nondeterminism of the program is controlled by the guard mechanism. After the first clause is committed, the subsequent goal has following form:

```
Z = [1|Out], merge([2], V, Out).
```

□

2.2 Partial Correctness

In GHC, the direction for execution of a program is indicated implicitly by the guard part. For the example of 'merge(X, Y, Z)' in the previous section, X and Y are for inputs and Z is for output. In the case of defining partial correctness, the explicit description of the execution direction of programs is useful. The description of execution direction of programs is expressed by annotating each argument in the head parts with + (input argument) or - (output argument). For a program with annotated head parts, an invocation of a clause C by a goal G is consistent with the annotation if C is invoked by G with variable terms in all arguments annotated with -. For example, when the 'append' program is annotated as:

```
append(X+, Y+, Z-) :- X = [] | Y = Z.
append(X+, Y+, Z-) :- X = [A|X1] | append(X1, Y, Z1), Z = [A|Z1].
```

then 'append([1,2],[3,4],X)' and 'append([a],U,V)' are consistent with the above annotation, and 'append([1,2,3],W,[1,2,3,4,5])' is not.

[Def. 1]
Let D be a set of guarded clauses:

```
H1:- B11,…, B1n1 | A11, …, A1m1.
H2:- B12,…, B1n2 | A12, …, A1m2.
                 :
                 :
Hs:- B1s,…, B1ns | A1s, …, A1ms.
```

Each argument of H1, …,Hs is annotated by + or -.

Let G1, …, Gk be goals, Φ and Ψ be input and output condition respectively. G1, …, Gk is partially correct wrt. Φ and Ψ iff for any computation with input satisfying Φ, following (1) or (2) is true.

(1) G1, …, Gk do not succeed or at least one invocation which is not consistent with the annotation in H1, …,Hs is called during the computation.

(2) Ψ is true for the result of the computation.

The above situation is denoted as :

$$\Phi\ \{G1,\ \cdots,\ Gk\}_{D}\ \Psi.$$

D after } is abbreviated if there is no confusion. In the above definition, the notions of 'computation' and 'success' are used. Strictly speaking, these notions should be defined based on operational semantics of GHC (Takeuchi 86).

(Example)

For following GHC program:

```
reverse(X+, Y-) :- true | rev1(X, [], Y).
rev1(E+, Y+, Z-) :- E = [] | Y = Z.
rev1(C+, Y+, Z-) :- C = [A|Y] | rev1(X, [A|Y], Z).
```

'reverse(U, V), reverse(V,W)' is partially correct w.r.t. input condition 'true' and output condition 'U = W'. In other words:

```
true {reverse(U, V), reverse(V, W)} U = W.
```

□

2.3 Inference Rules

This section shows an elementary fragment of the set of inference rules. Variables appearing in a proof are abstracting a number of terms in TERM which appear during the execution of a program. In this article, 'variable' means abstract variables appearing in a proof, which are denoted by lower case letters u, v, z1, z2, … 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, …. Variables appearing in program clauses are considered as variables when they are used for verification. 'Term' means an element of TERM. A term appearing in a proof is called 'term form'.

〈Substitution〉

$$\frac{\Phi\ \{G1, \cdots, Gn\}\ \Psi}{\sigma\Phi\ \{\sigma G1, \cdots, \sigma Gn\}\ \sigma\Psi}$$

where σ is a substitution.

〈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 $P1, \cdots, Ps$ is the sequence of all formulas defined as following. There is a guarded clause:

$$Hj :- Bj1, \cdots, Bj\ h_j \mid Aj1, \cdots, Aj\ m_j. \quad (1 \leqq j \leqq s)$$

in D such that Hj is unifiable with G ($\sigma_j G = \sigma_j Hj$ for some substitution σj),
and σ_j does not substitute any variable appearing in Φ in arguments of Hj annotated with '-', and Pj has a following form:

$$Pj \equiv \bigwedge_{k=1,\ h_j} \sigma_j Bjk \wedge \sigma_j \Phi\ \{\sigma_j Aj1, \cdots, \sigma_j Ajm\}\ \sigma_j \Psi$$

where any variable appearing in the term form which is unified with a term form appearing in an argument annotated - does not appear at the left of ' {'.

〈Parallel〉

$$\frac{\Phi 1\ \{G1\}\ \Psi 1, \cdots, \Phi n\ \{Gn\}\ \Psi n}{\bigwedge_{i=1,n} \Phi i\ \{G1, \cdots, Gn\} \bigwedge_{i=1,n} \Psi i}$$

where Φi and Φj (Ψi and Ψj) do not share any variable for all i, j ($1 \leqq i, j \leqq n,\ i \neq j$).

In this system, all formula which are true in the domain of the program are regarded as an axiom like usual Hoare-like system.

4 Proof Schema

In most of Hoare like proof system, 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. Such definition of proof schema is found in [Murakami 86].

[Def. 2]
A proof schema of formula $\Phi\ \{ G1, \cdots, Gn \}\ \Psi$ is a tree such that:

1) The root of the tree corresponds to Φ { G1, ···, Gn } Ψ .

2) For every node n , one of a) or b) following is true.

a) For some inference rule (shown in 2.3), n is an instance of a conclusion and each child of n corresponds to a premise.

b) n is a leaf and one of following is true,

(i) n is an axiom. In other words, n is a theorem without {, } -part.
(ii) n is identical to one of its ancestors n' and Derivation 2) rule is used at least once on the path form n' to n.

2.5 Example of Proof

This is a program to find the maximum element from a binary tree. For example, the result of 'max([[[1],[4]],[3]], X)' is X = 4.

```
max([x, y]+, z-) :- y ≠ [] | max(x, z1), max(y, z2), max2(z1, z2, z).
max([x]+, z-) :- atom(x) | z = x.
```

Now the proof of following formula is described:

$$\text{binary}(u)\ \{\ \max(u, v)\ \}\ \ v \in |u| \wedge (\forall x \in |u| \Rightarrow v \geqq x) \quad \text{---------- (0)}$$

where 'binary(u)' means that u is a binary tree of [A]. 'max2(z1, z2, z)' is a predicate that takes z1 and z2 as input and returns the greater element of them as z. The definition and correctness proof of this predicate are omitted. |u| means the set of elements of list 'u'.

It is easy to show that:

$$\text{binary}([y1]) \wedge \text{atom}(y1) \Rightarrow (\forall x \in [y1] \Rightarrow y1 \geqq x). \quad \text{---------- (1)}$$

From (1) :

$$\text{binary}([y1]) \wedge \text{atom}(y1) \Rightarrow (m = y1 \Rightarrow (\forall x \in [y1] \Rightarrow m \geqq x)). \quad \text{---------- (2)}$$

Applying the rule ⟨Derivation 1⟩ :

$$\text{binary}([y1]) \wedge \text{atom}(y1)\ \{\ m = y1\ \}\ (\forall x \in \{y1\} \Rightarrow m \geqq x). \quad \text{------- (3)}$$

For 'max2', it is easy to show:

$$\text{true}\ \{\ \max2(z1', z2', z3')\ \}\ ((z1' = z3' \wedge z1' \geqq z2') \vee (z2' = z3' \wedge z2' \geqq z1')). \quad \text{----- (4)}$$

On the other hand, applying substitutions $\sigma 1$ = { u1/u, v1/v } and $\sigma 2$ = { u2/u, v2/v } to (0):

$$\text{binary}(u1)\ \{\ \max(u1, v1)\ \}\ v1 \in |u1| \wedge (\forall x \in |u1| \Rightarrow v1 \geqq x) \quad \text{--------- (5)}$$

$$\text{binary}(u2)\ \{\ \max(u2, v2)\ \}\ v2 \in |u2| \wedge (\forall x \in |u2| \Rightarrow v2 \geqq x). \quad \text{--------- (6)}$$

From (4), (5), (6) and the rule ⟨Parallel⟩ :

$$\text{true} \wedge \text{binary}(u1) \wedge \text{binary}(u2)$$
$$\{\ \max(u1, v1),\ \max(u2, v2),\ \max2(z1', z2', z3')\ \}$$
$$v1 \in |u1| \wedge (\forall x \in |u1| \Rightarrow v1 \geqq x) \wedge$$
$$v2 \in |u2| \wedge (\forall x \in |u2| \Rightarrow v2 \geqq x) \wedge$$
$$((z1' = z3' \wedge z1' \geqq z2') \vee$$
$$(z2' = z3' \wedge z2' \geqq z1')).$$ ---------- (7)

Applying ⟨Substitution⟩ to (7) with $\sigma 3 = \{\ v1/z1',\ v2/z2',\ m/z3'\ \}$:

$$\text{true} \wedge \text{binary}(u1) \wedge \text{binary}(u2)$$
$$\{\ \max(u1, v1),\ \max(u2, v2),\ \max2(v1, v2, m)\ \}$$
$$v1 \in |u1| \wedge (\forall x \in |u1| \Rightarrow v1 \geqq x) \wedge$$
$$v2 \in |u2| \wedge (\forall x \in |u2| \Rightarrow v2 \geqq x) \wedge$$
$$((v1 = m \wedge v1 \geqq v2) \vee$$
$$(v2 = m \wedge v2 \geqq v1)).$$ ---------- (8)

(9) and (10) can easily be shown from the properties of binary trees.

$$v1 \in |u1| \wedge (\forall x \in |u1| \rightarrow v1 \geqq x) \wedge$$
$$v2 \in |u2| \wedge (\forall x \in |u2| \Rightarrow v2 \geqq x) \wedge$$
$$((v1 = m \wedge v1 \geqq v2) \vee$$
$$(v2 = m \wedge v2 \geqq v1))$$
$$\Rightarrow\ m \in |\{u1, u2\}| \wedge (\forall x \in |\{u1, u2\}| \Rightarrow m \geqq x)$$ ---------- (9)

$$\text{binary}([u1, u2]) \Rightarrow \text{binary}(u1) \wedge \text{binary}(u2)$$ ---------- (10)

Applying ⟨Consequence 1⟩ and ⟨Consequence 2⟩ to (8), (9), (10) :

$$\text{binary}([u1, u2])\ \{\ \max(u1, v1),\ \max(u2, v2),\ \max2(v1, v2, m)\ \}$$
$$m \in |\{u1, u2\}| \wedge (\forall x \in |\{u1, u2\}| \Rightarrow m \geqq x).$$ ---------- (11)

(12) is trivial.

$$\text{binary}([u1, u2]) \wedge u2 \neq \{\} \Rightarrow \text{binary}([u1, u2])$$ ------ (12)

From (11) and (12), applying ⟨Consequence 2⟩ :

$$\text{binary}([u1, u2]) \wedge u2 \neq \{\}$$
$$\{\ \max(u1, v1),\ \max(u2, v2),\ \max2(v1, v2, m)\ \}$$
$$m \in |\{u1, u2\}| \wedge (\forall x \in |\{u1, u2\}| \Rightarrow m \geqq x).$$ ---------- (13)

From (3) and (13), applying ⟨Derivation 2⟩ :

$$\text{binary}(u)\ \{\ \max(u, v)\ \}\ \ v \in |u| \wedge (\forall x \in |u| \Rightarrow v \geqq x)$$

This formula is identical to (0). Thus the proof schema is constructed.
Fig. 1 shows the whole proof schema.

□

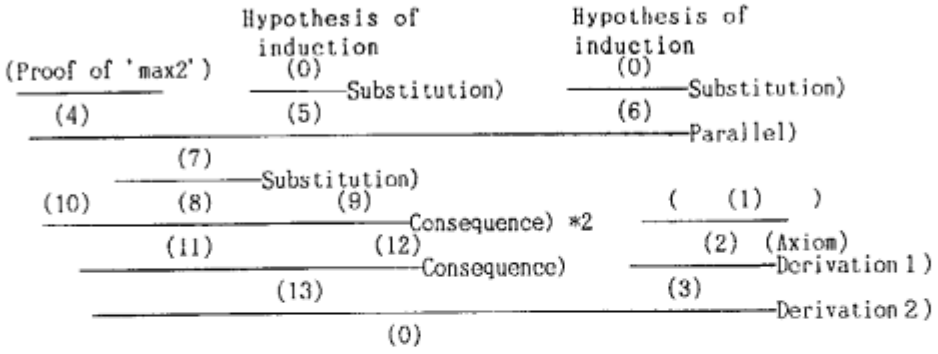

Fig 1

3 Control of nondeterminism with suspend

3.1 ↓ annotation

This section shows how to prove partial correctness of programs which contains control of nondeterminism with the guard mechanism. The following program is a fragment of Brock-Ackermann's anomaly in GHC [Takeuchi 86].

```
p2([u, v|x]+, w-) :- true | w = [u, v].

merge([u|x]+, y+, w-) :- true | w = [u|v], merge(x, y, v).
merge(x+, [u|y]+, w-) :- true | w = [u|v], merge(x, y, v).
merge(x+, []+, w-) :- true | x = w.
merge([]+, y+, w-) :- true | y = w.
```

In Brock-Ackermann's anomaly, 'merge' is invoked with a variable term in its second argument and executed with p2 in parallel. So when 'merge' is invoked, the first clause commits first in any case.

As in this example, it also often happens that only the invocation where some arguments are variable termsi s of interests. In particular, it is essential when discussing the effect of synchronization. In this section, the situation that some variables are variable terms is described by annotating the variables with ↓. For example, the invocation of 'merge' with the variable term in the second argument is described as:

```
merge(u, v↓, w).
```

The notion of partial correctness is redefined as follows for goals containing a variable annotated with ↓.

[Def. 3]
Let D be a set of guarded clauses as [Def. 1]. Let G1, …, Gk be goals, and Φ and Ψ be the input and output conditions. G1, …, Gk is partially correct w.r.t. Φ and Ψ iff for any computation starting with variable term in variable annotated with ↓ and input satisfying Φ, (1), or (2) is true.

(1) G1, …, Gk do not succeed or at least one invocation which is not

consistent with the annotation in H1, ···, Hs is called during the computation.

(2) Ψ is true for the result of the computation.

New inference rules for the formula containing variables annotated with ↓ are introduced in the following section. When variable x is annotated at some occurrence in a formula, x is annotated at all occurrence in the formula. The result of unification of x and y ↓ is y ↓.

3.2 Inference Rules for Formulas with ↓ Annotation

The following rules are introduced instead of ⟨Substitution⟩ and ⟨Derivation 2⟩ in Section 2.3.

⟨Substitution'⟩

$$\frac{\Phi\ (G1,\ \cdots,\ Gn)\ \Psi}{\sigma\Phi\ (\sigma G1,\cdots,\ \sigma Gn)\ \sigma\Psi}$$

where σ does not substitute a non-variable term form for the variable annotated with ↓, and does not make variables annotated with ↓ and variables appearing in G1, ···, Gn which are not annotated, identical.

⟨Derivation 2'⟩

$$\frac{P1,\ \cdots,\ Ps}{\Phi\ (G)\ \Psi}$$

where P1, ···, Ps is the sequence of all Pj ($1 \leqq j \leqq s$) defined as follows:

There is a guarded clause:

$$Hj\ :-\ Bj1,\ \cdots,\ Bj\ h_j\ \mid Aj1,\ \cdots,\ Aj\ m_j. \qquad (j = 1,\ s)$$

in D such that Hj is unifiable with G ($\sigma_j G = \sigma_j Hj$), σ_j does not substitute any variable appearing in Φ for variables in arguments of Hj annotated with -, does not substitute non-variable term form for the variable annotated with ↓ in the unification of a term appearing in G, and a term form appears in Hj as an argument annotated with +, and Pj has the following form:

$$Pj \equiv \bigwedge_{k=1,\ h_j} \sigma_j Bjk \wedge \sigma_j \Phi\ (\sigma_j Aj1,\ \cdots,\ \sigma_j Ajm)\ \sigma_j \Psi$$

where any variable appearing in the term form which unified with an argument annotated - does not appear at the left of (.

A formula containing variables annotated with ↓ can be inferred by following rules:

1) If x is annotated in the premises then x is also annotated in the conclusion.

2) A variable x appearing in G of Φ {G} Ψ can be annotated with $\downarrow$ if Φ {G} Ψ is a conclusion of ⟨Derivation 2'⟩ and for any j ($1 \leqq j \leqq s$) σ_j does not substitute a non-variable term form for x.

3) A variable x appearing in $S = T$ of Φ {$S = T$} Ψ can be annotated if Φ {$S = T$} Ψ is a conclusion of ⟨Derivation 1⟩.

4) A variable x appearing in G of Φ {G} Ψ can be annotated if Φ {G} Ψ appears in a proof schema as a hypothesis of induction.

Using this refinement of the axiom system, the following can be proved.

[a, b] = u {merge(u, v↓, w), p2(w, o)} o = [a, b]

3.2 Example of proof 2

In this section, the proof of following formula is given.

[A, B] = U {merge(U, V↓, W), p2(W, Out)} Out = [A, B]

From:

$$\text{true} \wedge V = O3 \Rightarrow V = O3 \quad \text{---------- (1)}$$

applying ⟨Derivation 1⟩:

$$\text{true}\ \{V\downarrow = O3\}\ V = O3\ . \quad \text{---------- (2)}$$

On the other hand, from:

$$\text{true} \wedge O1 = [B1|O2] \Rightarrow O1 = [B1|O2] \quad \text{----------(3)}$$

applying ⟨Derivation 1⟩:

$$\text{true}\ \{O1 = [B1|O2]\}\ O1 = [B1|O2]. \quad \text{---------- (4)}$$

From (2), applying ⟨Derivation 2'⟩:

$$\text{true}\ \{\ \text{merge}([], V\downarrow, O3)\ \}\ V = O3. \quad \text{---------- (5)}$$

From (4) and (5), applying ⟨Parallel⟩:

$$\text{true}\ \{O1 = [B1|O2], \text{merge}([], V\downarrow, O3)\ \}\ V = O3 \wedge O1 = [B1|O2]. \quad \text{---------- (6)}$$

Applying ⟨Substitution'⟩ with $\sigma = \{O2 / O3\}$:

$$\text{true}\ \{O1 = [B1|O2], \text{merge}([], V\downarrow, O2)\ \}\ V = O3 \wedge O1 = [B1|O2]. \quad \text{----------(7)}$$

From:

$$[B1] = [B1] \Rightarrow \text{true},\quad V = O3 \wedge O1 = [B1|O2] \Rightarrow O1 = [B1|V] \quad \text{----------(8)}$$

applying ⟨Consequence 1⟩ and ⟨Consequence 2⟩:

$$[B1] = [B1]\ \{O1 = [B1|O2], \text{merge}([], V\downarrow, O2)\ \}\ O1 = [B1|V]. \quad \text{---------- (9)}$$

On the other hand, from:

$$[B1] = [] \wedge V = O1 \Rightarrow O1 = [B1|V] \quad \text{----------(10)}$$

applying ⟨Derivation 1⟩:

$$[B1] = [\,]\ \{V\downarrow = O1\ \}\ O1 = [B1|V]. \quad \text{---------- (11)}$$

From (9) and (11), applying ⟨Derivation 2'⟩:

$$[B1] = I1\ \{merge(I1, V\downarrow, O1)\}\ O1 = [B1|V] \quad \text{---------- (12)}$$

where σ_3 for (9) is :

$$\{[B1]/I1,\ B1/A,\ [\,]/Ix,\ V/Iy,\ O1/O,\ O2/Out\ \},$$

and σ_3 for (11) is:

$$\{[\,]/I1,\ V/Iy,\ O1/O\ \}.$$

Furthermore from:

$$true \wedge W = [A1|O11] \Rightarrow (O11 = [B11|V1] \Rightarrow W = [A1, B11|V1])$$

----------(13)

applying ⟨Derivation 1⟩:

$$true\ \{\ W = [A1|O11]\ \}\ (O11 = [B11|V1] \Rightarrow W = [A1, B11|V1]).$$

---------- (14)

From (12) and (14), applying ⟨Parallel⟩:

$$[B1] = I1\ \{W = [A1|O11],\ merge(I1, V\downarrow, O1)\}\ O1 = [B1|V] \wedge (O11 = [B11|V1] \Rightarrow W = [A1, B11|V1])$$

----------(15)

Applying ⟨Substitution⟩ with $\sigma' = \{O1/O11,\ B1/B11,\ V/V1\}$:

$$[B1] = I1\ \{W = [A1|O1],\ merge(I1, V\downarrow, O1)\}\ O1 = [B1|V] \wedge (O1 = [B1|V] \Rightarrow W = [A1, B1|V]).$$

----------(16)

From:

$$O1 = [B1|V] \wedge (O1 = [B1|V] \Rightarrow W = [A1, B1|V]) \Rightarrow W = [A1, B1|V]$$

$$[A1, B1] = [A1|I1] \Rightarrow [B1] = I1$$

----------(17)

applying ⟨Consequence 1⟩ and ⟨Consequence 2⟩:

$$[A1, B1] = [A1|I1]\ \{W = [A1|O1],\ merge(I1, V\downarrow, O1)\}\ W = [A1, B1|V].$$

---------- (18)

On the other hand from:

$$[A1, B1] = [\,] \wedge V = W \Rightarrow W = [A1, B1|V]$$

----------(19)

applying ⟨Derivation 1⟩:

$$[A1, B1] = [\,]\ \ (V\downarrow = W\ )\ \ W = [A1, B1|V]. \quad \text{-------- (20)}$$

From (9) and (10), applying ⟨Derivation 2'⟩:

$$[A1, B1] = I2\ \{merge(I2, V\downarrow, W)\}\ W = [A1, B1|V] \quad \text{---------- (21)}$$

where σ_3 for (18) is:

$$\{ [A1|I1)/I2,\ A1/A,\ I1/Ix,\ V/Iy,\ W/O,\ O1/Out \}$$

and for (20) is:

$$\{()/I2,\ V/Ix,\ W/O \}\ .$$

On the other hand, the proof of the correctness of p2 is as following. At first, it is easy to show:

$$true \wedge Out1 = [A4, B4] \Rightarrow ([A4, B4|\_] = [A3, B3|V2] \Rightarrow Out1 = [A3, B3]).$$

----------(22)

Applying ⟨Derivation 1⟩:

$$true\ (Out1 = [A4, B4])\ ([A4, B4|\_] = [A3, B3|V2] \Rightarrow Out1 = [A3, B3]).$$

----------(23)

Applying ⟨Derivation 2⟩:

$$true\ (p2(W1, Out1))\ W1 = [A3, B3|V2] \Rightarrow Out1 = [A3, B3].$$

---------- (24)

where:

$$\sigma = \{[A4, B4|\_]/W1,\ Out1/O,\ A4/A,\ B4/B\}.$$

From (21) and (24), applying ⟨Parallel⟩:

$$[A1, B1] = I2\ (merge(I2, V\downarrow, W),\ p2(W1, Out1))\ W = [A1, B1|V] \wedge (W1 = [A3, B3|V2] \rightarrow Out1 = [A3, B3]).$$

---------- (25)

Applying ⟨Substitution⟩

$$[A1, B1] = I2\ (merge(I2, V\downarrow, W),\ p2(W, Out))\ W = [A1, B1|V] \wedge (W = [A1, B1|V] \Rightarrow Out = [A1, B1]).$$

---------- (26)

From:

$$W = [A1, B1|V] \wedge (W = [A1, B1|V] \Rightarrow Out = [A1, B1]) \Rightarrow Out = [A1, B1]$$

----------(27)

applying ⟨Consequence 1⟩:

$$[A1, B1] = I2\ (merge(I2, V\downarrow, W),\ p2(W, Out))\ Out = [A1, B1]$$

----------(28)

The result is derived applying ⟨Substitution⟩ with

$$\sigma'' = \{A/A1,\ B/B1,\ U/I2\}\ .$$

$$[A, B] = U\ (merge(U, V\downarrow, W),\ p2(W, Out))\ Out = [A, B]$$

----------(29)

Fig. 2 shows the whole proof schema. □

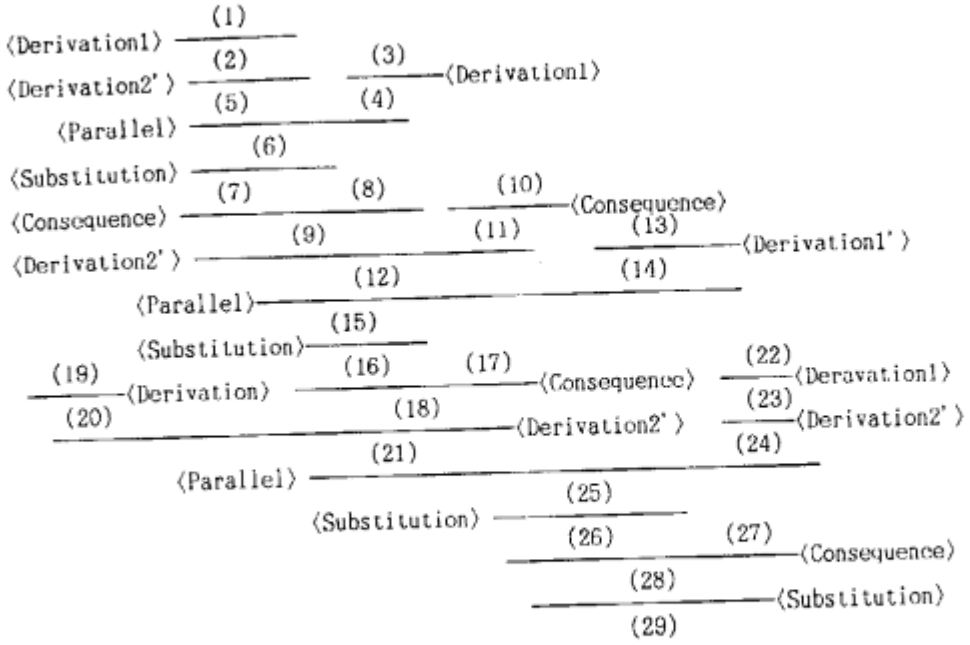

Fig.2

4. Conclusion

This abstract introduced fragments of an axiomatic system for proving partial correctness. These are the most elementary parts. It is not enough to prove partial correctness for stronger output condition. For example, in Block-Ackermann's anomaly, the following can be proved in the fragment system introduced here:

$$n = [a] \ \{t2(n, o)\} \ o = [a, a] \vee o = [a, s(a)].$$

However, the following cannot be proved in this system although this formula is true:

$$n = [a] \ \{t2(n, o)\} \ o = [a, a].$$

To prove a correctness property like this, a method to formalize effects of synchronization is needed. This is done by introducing a new notation representing communications between processes explicitly and refinement of the inference rule (Parallel). This refinement of axiomatic system and examples of the proof will appear in [Murakami 87].

Acknowledgements

I would like to thank Dr. K. Furukawa, Dr. K. Ueda, Dr. J. Tanaka and Mr. H. Seki, and all other members of First and Third Laboratories. of ICOT for many useful discussions.

[References]

[Clark 86] K. L. Clark and S. Gregory, PARLOG: Parallel programming in logic, ACM Trans. on Programming Language and Systems 86, 1986

(Deransart 87) P. Deransart, Partial Correctness of Logic Programs, submitted to Symp. on Logic Programming 87

(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 85) G. Levi and C. Palamidessi, The Declarative Semantics of Logical Read-only Variables. Prc. of Symp. on Logic Programming 85 1985

(Levi 87) G. Levi and C. Palamidessi, An Approach to The Declarative Semantics of Synchronization in Logic Language, to appear in Proc. of International Conf. on Logic Programming 87, 1987

(Murakami 86) M.Murakami and Y. Inagaki, Verification System for Partial Correctness of Communicating Sequential Processes, Systems and Computers in Japan, 1986

(Murakami 87) M. Murakami, Axiomatic Semantics of Guarded Horn Clauses, Tec. Rep. of ICOT in preparation

(Saraswat 85) V. A. Saraswat, Partiul Correctness Semantics for CP (↓,|,&), Lecture Notes in Comp. Sci., no. 206, 1985

(Saundararajan 84) N. Saundararajan, Axiomatic Semantics of Communicating Sequential Processes, ACM Trans. on Programming Languages and Systems, Vol. 6, No. 4, 1984

(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, Tec. 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, Tec. Memo of ICOT, TM-0160, 1986