ICOT Technical Report: TR-551

TR-551

A Fixpoint Semantics of Guarded Horn Clauses

by
T. Kanamori & M. Ueno (Mitsubishi)

April, 1990

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

A Fixpoint Semantics of Guarded Horn Clauses

Tadashi KANAMORI Machi UENO

Central Research Laboratory
Mitsubishi Electric Corporation
8-1-1, Tsukaguchi-Honmachi
Amagasaki, Hyogo, JAPAN 661

Abstract

This paper presents a fixpoint semantics of flat GHC programs. The framework is parallel to the fixpoint semantics of Prolog programs. First, instead of an atom and a Herbrand interpretation, an *atom behavior* and a *behavior interpretation* are considered, where an atom behavior is a finite set of atom pairs $(A\sigma, A\tau)$, and a behavior interpretation is a set of atom behaviors. Next, instead of the conjunction and the implication for atoms, the parallel conjunction and the guarded implication for atom behaviors are introduced. A transformation T for behavior interpretations associated with a GHC program is defined using those two notions. Then, the semantics of a GHC program is defined as the least fixpoint of the transformation T. The relations to an operational semantics of GHC programs and to the semantics of Prolog programs are discussed as well.

Keywords : Fixpoint Semantics, Operational Semantics, GHC.

Contents

1. Introduction

Guarded Horn Clauses (GHC), as well as Concurrent Prolog and Parlog, is a programming language originated from the attempts for designing Prolog-like languages suitable for parallel execution [31],[33]. As is well known, two remarkable features brought by the notion of guard distinguish GHC from Prolog.

One remarkable feature of GHC is the suspension (and synchronization) mechanism in the guards by prohibiting the instantiation of the variables appearing in caller goals. Because of this mechanism, the execution of a goal consisting of several atoms proceeds by importing and exporting the instantiation information, so that some goal can succeed when other goals co-exist, even if it never succeeds as a single goal. In general, some goal behaves in a quite different manner according to the behavior of co-existing goals, hence, if only the final form of the execution as a single goal is considered, the execution result of the goal is not necessarily synthesized from the execution results of the individual atoms in the goal (cf. Section 6).

The other remarkable feature of GHC is the committed-choice mechanism by throwing away alternative courses at passing the guards. Because of this mechanism, once we have found a clause to which the execution of a goal is committed, there occurs no backtracking. So, even if there exist several solutions to a goal, only one of them is obtained as its solution. Moreover, even the same initial goal with the same final form might succeed, fail or be suspended depending on to which clause the execution is committed.

To clarify the semantics of the (so called) parallel logic programs, which share the same features in greater or lesser degree, several attempts have been made [2],[11],[15],[16],[22], [24],[25],[26],[27],[28],[30],[32]. (Besides, several attempts for reasoning about GHC programs, e.g., verification [13],[23], transformation [7],[8],[9],[15],[35], and debugging [12],[17],[18],[20], [21],[29], have been made as well based on those semantics.) It has been sometimes said that the search incompleteness due to the second feature is the most important difference that makes the semantics of GHC programs extra-logical. The semantics in this paper is, however, concerned with *all the possible* computation, so that we will pay more attention to the first feature (and ignore the problems of each specific computation), just as the semantics of Prolog assumes appropriate nondeterminism (and ignores the backtracking mechanism) to consider all the possible successful computation.

This paper presents a fixpoint semantics of flat GHC programs. The framework is parallel to the fixpoint semantics of Prolog programs. First, instead of an atom and a Herbrand interpretation, an *atom behavior* and a *behavior interpretation* are considered, where an atom behavior is a set of pairs $(A\sigma, A\tau)$, and a *behavior interpretation* is a set of atom behaviors. Next, instead of the conjunction and the implication for atoms, the parallel conjunction and the guarded implication for atom behaviors are introduced. A transformation T for behavior interpretations associated with a GHC program is defined based on those two notions. Then, the semantics of a GHC program is defined as the least fixpoint of the associated transformation T. The relations to an operational semantics of GHC programs and to the semantics of Prolog programs are discussed as well.

The rest of this paper is organized as follows: Section 2 explains flat GHC programs and their execution. Section 3 introduces an operational semantics of flat GHC programs based on the notion of computation tree, and Section 4 a fixpoint semantics based on the notions of atom behavior, behavior interpretation and transformation of behavior interpretations. Then Section 5 shows the equivalence of those two semantics. Section 6 gives an explanation to the famous Brock-Ackerman's anomaly. Last, Section 7 compares our semantics with that

of Prolog programs, and Section 8 discusses the problem of the truly parallel execution and the non-deterministic sequential execution.

2. Flat GHC

This section explains GHC mostly following the explanation of Ueda [34]. Symbols beginning with uppercase letters are used for variables, and ones beginning with lower case letters for constant, function and predicate symbols, following the syntactic convention of DECsystem10 Prolog [3].

(1) Program

A *clause* is an expression of the form:

$$H :- G_1 , G_2 , \ldots , G_m \mid B_1 , B_2 , \ldots , B_n \qquad (m, n \geq 0),$$

where H, G_i's and B_j's are atoms ($1 \leq i \leq m, 1 \leq j \leq n$). H is called a *clause head*, the G_i's are called *guard atoms*, and the B_j's are called *body atoms*. The symbol "|" is called a *commitment operator*. The part of a clause before "|" is called a *guard*, and the part after "|" is called a *body*. (When $m = n = 0$, ":-" and "|" are omitted. Note that the clause head is included in the guard.)

One primitive (infix) binary predicate "=" for unifying two terms is predefined by the language. Other primitive predicates are predefined using a (possibly infinite) set of clauses such that

- each clause is of the form "$H :- |B_1, B_2, \ldots, B_n$" $(n \geq 0)$,
- H is not unifiable with the head of any other clause in the set, and
- each body atom B_j is an equation of the form "$s_j = t_j$" $(1 \leq j \leq n)$.

The primitive predicates used in practice are not excluded by this condition. Atoms with primitive predicates are called *primitive atoms*.

A clause is called a *flat clause* when each guard atom G_i is a primitive atom $(1 \leq i \leq m)$. A *program* is a finite set of flat clauses.

Example 2.1 Let P be the set of the following flat clauses:

```
C01: p1(X,Y,Z) :- |
         double(X,XX), double(Y,YY), merge(XX,YY,W), one-by-one(W,Z).
C02: p2(X,Y,Z) :- |
         double(X,XX), double(Y,YY), merge(XX,YY,W), two-at-once(W,Z).
C03: double(0,AA) :- | AA=[0,0].
C04: double(1,AA) :- | AA=[1,1].
C05: merge([A|Xs],Ys,Zs) :- | Zs=[A|Zs1], merge(Xs,Ys,Zs1).
C06: merge(Xs,[A|Ys],Zs) :- | Zs=[A|Zs1], merge(Xs,Ys,Zs1).
C07: merge([ ],Ys,Zs) :- | Zs=Ys.
C08: merge(Xs,[ ],Zs) :- | Zs=Xs.
C09: one-by-one([A|W],Z) :- | Z=[A|Z1], next-one(W,Z1).
C10: next-one([B|W],Z) :- | Z=[B].
C11: two-at-once([A,B|W],Z) :- | Z=[A|Z1], Z1=[B].
C12: complement([0|Z],Y) :- | Y=1.
C13: complement([1|Z],Y) :- | Y=0.
```

Then, P is a program. Here, "*double*" is a predicate to duplicate its input element (either 0 or 1) and make a list consisting of the two elements, and "*merge*" is the predicate for non-deterministically merging two input streams into one output stream. The only difference between "*p1*" and "*p2*" is that "*one-by-one*" invoked from "*p1*" pulls out first two elements from the merged stream one by one as soon as each element appears in the merged stream, while "*two-at-once*" invoked from "*p2*" pulls out first two elements from the merged stream only when two elements appear in the merged stream. "*complement*" is a predicate to return the complement of the head element 0 or 1. (This is a slightly modified version of the program in [4]).

(2) Goal

A *goal* is an expression of the form:

$?- A_1, A_2, \ldots, A_k \qquad (k \geq 0).$

A goal is called an *empty goal* when k is equal to 0.

Example 2.2 The following are GHC goals.

```
?- p1(0,Y,Z), complement(Z,Y).
?- p2(0,Y,Z), complement(Z,Y).
```

(3) Execution

The execution of a GHC goal with respect to a given GHC program tries to solve the goal, i.e., reduce the goal to the empty goal, using the clauses in the GHC program in the same way as Prolog but possibly a fully parallel manner provided that the following "rules of suspension" and "rule of commitment" are observed.

Rules of Suspension

(a) Unification invoked directly or indirectly in the guard of a clause C called by an atom G (i.e., unification of G with the head of C and any unification invoked by solving the guard atoms of C) cannot instantiate the atom G.

(b) Unification invoked directly or indirectly in the body of a clause C called by an atom G cannot instantiate the guard of C or G until C is selected for commitment (see below).

A piece of unification that can succeed only by causing such instantiation is suspended until it can succeed without causing such instantiation.

Rule of Commitment

When some clause C called by an atom G succeeds in solving its guard, that clause C tries to be selected for subsequent computation of G. To be selected, C must first confirm that no other clause in the program have been selected for G. If confirmed, C is selected indivisibly, and the execution of G is said to be *committed* to the clause C.

Example 2.3 The execution of goal

```
?- p1(0,Y,Z), complement(Z,Y)
```

in P succeeds with answers

```
p1(0,1,[0,0]),complement([0,0],1),
p1(0,1,[0,1]),complement([0,1],1),
```

while the execution of goal

```
?- p2(0,Y,Z), complement(Z,Y)
```
in P succeeds only with answer
```
p1(0,1,[0,0]),complement([0,0],1).
```

(4) Success, Failure and Suspension

Let A be an atom and C be a clause called by A. When the guard of C is solved with answer substitution, say θ, for the variables appearing in the guard of C without instantiating A, then the execution of A is said to *succeed in the guard of clause C with substitution θ*. Otherwise, the execution of A is said to *be suspended in the guard of clause C*. (The latter case includes two cases. One is the case when the unification invoked, either directly or indirectly, in the guard of C instantiates the atom A. The other is the case when the guard cannot be solved even if the instantiation of A is permitted. We will not make a distiction between them hereafter.)

Note that, due to the restriction on the primitive predicates, the execution of atom A *can succeed* in the guard of a clause C with substitution θ by committing to appropriate clauses in the guard, if and only if the execution of atom A *does succeed* in the guard of C with θ by committing to any committable clauses in the guard. Similarly, the execution of goal A *can be suspended* in the guard of clause C, if and only if the execution of goal A *is suspended* in the guard of clause C.

An atom A is said to *succeed immediately in program P* when

- A is an equation for two unifiable terms, or
- there exists a clause C with no body atoms in P such that the execution of A succeeds in the guard of C.

An atom A is said to *be suspended immediately in program P* when the execution of A is suspended in the guard of any clause in P. An atom A is said to *fail immediately in program P* when A is an equation for two non-unifiable terms.

Notice the difference between the suspension in the guard of a specific clause and the immediate suspension in a program. Note also that the execution of non-primitive atoms can be still non-deterministic, though the execution of primitive atoms is deterministic.

Example 2.4 Let A be an atom of the form

$merge([0,0],Y,W)$.

Then, the execution of A succeeds in the guard of clause C_{05}, while the execution of A is suspended in the guards of clauses C_{06}, C_{07} and C_{08}.

Let B be an atom of the form

$two\text{-}at\text{-}once([0|W1],Z)$.

Then, the execution of B is suspended in program P.

3. An Operational Semantics of Flat GHC Programs

This section introduces an operational semantics of flat GHC programs. First, the actual computation process of an atom is represented by a tree, called a *computation tree*, based on the non-deterministic sequential execution. Next, a more abstracted aspect of the computation process is represented by a set of atom pairs, called an *atom behavior*. Then, the operational semantics of a flat GHC program is represented by a set of atom behaviors. (Note that, though we will use the non-deterministic sequential execution to formalize the necessary notions, most of the final notions formalized are independent of the sequentiality. See Section 8 for the details.)

The following sections assume familiarity with the basic terminology of first order logic, such as term, atom (atomic formula), formula and so on. Syntactic variables are X, Y, Z for variables; s, t for terms; C for clauses, possibly with primes and subscripts. "$\equiv$" is used to denote the syntactical identity of two expressions.

A *substitution* is defined as usual, and denoted by

$$< X_1 \Leftarrow t_1, X_2 \Leftarrow t_2, \ldots, X_l \Leftarrow t_l >,$$

where $X_1, X_2, \ldots, X_l$ are distinct variables. The set of variables $\{X_1, X_2, \ldots, X_l\}$ is called the *domain*, and the set of variables appearing in $t_1, t_2, \ldots, t_l$ is called the *range* of the substitution. A substitution is called a *renaming substitution* when it assigns a distinct variable to each variable. Substitutions are denoted by $\sigma, \tau, \mu, \nu, \theta, \eta$, and the empty substitution is denoted by $<>$.

An *atom* is defined as usual. Atoms are denoted by A, B, possibly with primes and subscripts. Two atoms are considered identical when they are identical up to renaming of the variables appearing in the atoms. An atom A is said to be *less instantiated than or equal to* atom B, and denoted by $A \leq B$, when there exists a substitution θ such that $A\theta$ is identical to B. An atom A is said to be *less instantiated than* atom B, and denoted by $A < B$, when $A \leq B$ and $B \not\leq A$.

A *goal* is a multiset of atoms. Goals are denoted by Γ, Δ, possibly with primes and subscripts. Two goals are considered identical when they are identical up to renaming of variables appearing in the goals. The relations $\leq$ and $<$ between goals are defined in the same way as those between atoms.

3.1 Computation Tree

In this section, we will introduce the notion of *computation tree*. (Cf. [14].)

(1) Labelled Tree

A computation tree is a special labelled tree. Hereafter, we will assume that

- each clause in program P is assigned a distinct clause identifier C_i $(i > 0)$,
- a unit clause "$X = X$" (not in program P) is assigned a clause identifier C_0, and
- a special clause identifier $C_?$ is prepared.

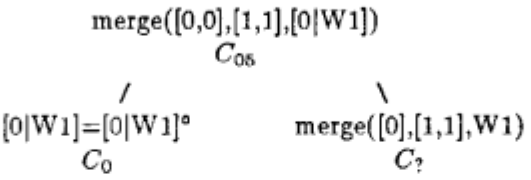
Definition Labelled Tree

A tree $\mathcal{T}$ is called a *labelled tree* when

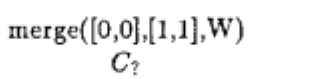
- the nodes of $\mathcal{T}$ are labelled with pairs of an atom and a clause identifier, and
- the terminal nodes of $\mathcal{T}$ are marked "success," "failure" or unmarked.

The atom part of the root node label of $\mathcal{T}$ is called the *root atom of* $\mathcal{T}$. Labelled trees are denoted by $\mathcal{T}$, possibly with primes and subscripts. Two labelled trees are considered identical when they are identical up to renaming of the variables appearing in the labels.

Example 3.1.1 $\mathcal{T}_2$ below is a labelled tree. The superscript "$\circ$" denotes the "success" mark. (The failure marks are denoted by superscripts "$\bullet$.")



$\mathcal{T}_0$ below, consisting of only one root node, is also a labelled tree.



(2) Extension of Labelled Trees

By modelling the non-deterministic sequential GHC execution, the extension of labelled trees is defined as follows:

Definition Immediate Extension of Labelled Trees

Let $\mathcal{T}$ and $\mathcal{T}'$ be labelled trees. Then, $\mathcal{T}'$ is called an *immediate extension of* $\mathcal{T}$ in program P when $\mathcal{T}'$ is obtained from $\mathcal{T}$ by the following operation:

Case 1 : When there exists an unmarked terminal node v in $\mathcal{T}$ labelled with $(s = t, C_?)$,

- replace the label of the node with $(s = t, C_0)$.

When s and t are unifiable, say by m.g.u. θ,

- modify the label (A', C') of each node in $\mathcal{T}$ to $(A'\theta, C')$, and
- mark the node v "success."

In this case, $\mathcal{T}'$ is called an immediate extension of $\mathcal{T}$ *with substitution* θ. When s and t are not unifiable,

- mark the node v "failure."

In this case, $\mathcal{T}'$ is called an immediate extension of $\mathcal{T}$ *with substitution* <>.

Case 2 : When there exist an unmarked terminal node v in $\mathcal{T}$ labelled with $(A, C_?)$ and a flat GHC clause C in P of the form

$$\mathrm{H} \text{ :- } \mathrm{G}_1, \mathrm{G}_2, \ldots, \mathrm{G}_m \mid \mathrm{B}_1, \mathrm{B}_2, \ldots, \mathrm{B}_n. \qquad (m, n \geq 0)$$

such that the execution of A succeeds in its guard with substitution θ, then let η be an instantiation of H to A, and

- replace the label of the node v with (A, C),
- add n child nodes of v labelled with $(B_1\eta\theta, C_?), (B_2\eta\theta, C_?), \ldots, (B_n\eta\theta, C_?)$ to v, and
- if $n = 0$, mark the node v "success."

In this case, $\mathcal{T}'$ is called an immediate extension of $\mathcal{T}$ *with substitution* θ.

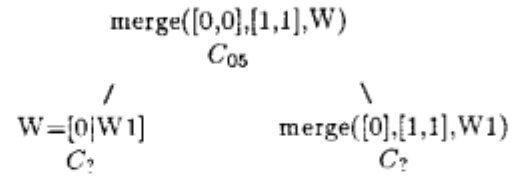
Definition Extension of Labelled Tree

Let $\mathcal{T}$ and $\mathcal{T}'$ be labelled trees. Then, $\mathcal{T}'$ is called an *extension* of $\mathcal{T}$ with substitution σ in program P when there are labelled trees $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k$ $(k \geq 0)$ such that

- $\mathcal{T}_0$ is $\mathcal{T}$,
- $\mathcal{T}_i$ is an immediate extension of $\mathcal{T}_{i-1}$ with substitution θ_i in P for $i = 1, 2 \ldots, k$,
- $\mathcal{T}_k$ is $\mathcal{T}'$, and
- σ is $\theta_1\theta_2 \cdots \theta_k$.

In particular, $\mathcal{T}'$ is called a *proper extension* of $\mathcal{T}$ when $k > 0$.

Example 3.1.2 Labelled tree $\mathcal{T}_1$ below is an immediate extension of the labelled tree $\mathcal{T}_0$ of Example 3.1.1 in the flat GHC program P.



The labelled tree $\mathcal{T}_2$ of Example 3.1.1 is an immediate extension of $\mathcal{T}_1$ in P, so that both $\mathcal{T}_1$ and $\mathcal{T}_2$ are extensions of $\mathcal{T}_0$ in P.

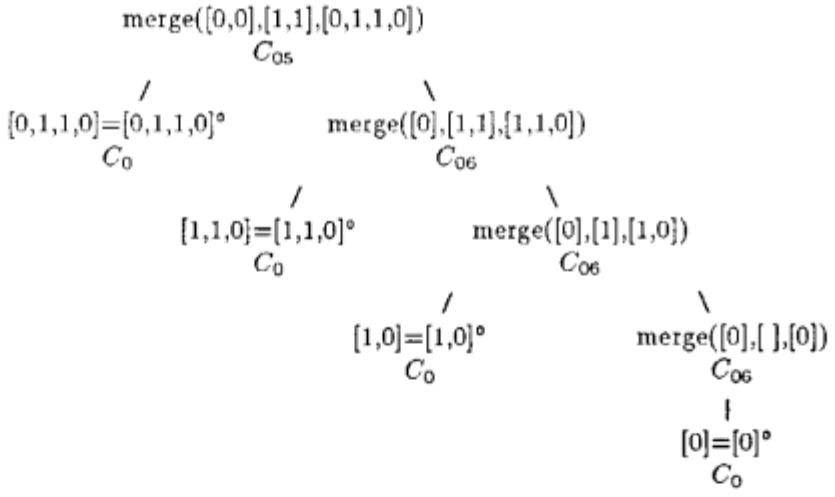
(3) Maximal Labelled Tree

Extending a labelled tree as far as possible corresponds to applying GHC execution as far as possible.

Definition Maximal Labelled Tree

A labelled tree $\mathcal{T}$ is called a *maximal labelled tree* in program P when there exists no proper extension of $\mathcal{T}$ in P.

Example 3.1.3 The labelled tree $\mathcal{T}_8$ below is a maximal labelled tree.



(4) Computation Tree

A computation tree models the GHC execution applied to an atom.

Definition Initial Tree

A labelled tree is called the *initial tree* of atom A when it consists of a single unmarked node labelled with $(A,C_?)$.

Definition Computation Tree

A labelled tree is called a *computation tree* of atom A with solution $A\theta$ in program P when it is an extension of the initial tree of A in P with substitution θ.

Example 3.1.4 The labelled tree $\mathcal{T}_0$ of Example 3.1.1 is an initial tree, so that $\mathcal{T}_1$ and $\mathcal{T}_2$ are computation trees in P. The labelled tree S_0 below is an initial tree.

merge([0,0],YY,W)
C_{05}

Hence, $\mathcal{S}_2$ below is a computation tree in P.

merge([0,0],YY,[0|W1])
C_{05}

/ \

[0|W1]=[0|W1]° merge([0],YY,W1)
C_0 $C_?$

(5) Success Tree, Failure Tree and Suspension Tree

Depending on how the terminal nodes are marked, maximal computation trees are classified into *success tree*, *failure tree* and *suspension tree*.

Definition Success Tree, Failure Tree and Suspension Tree
A maximal computation tree $\mathcal{T}$ in program P is called
- a *success tree* in P when all the terminal nodes in $\mathcal{T}$ are marked "success,"
- a *failure tree* in P when some terminal node in $\mathcal{T}$ is marked "failure,"
- a *suspension tree* in P otherwise.

A maximal computation tree is said to *have an uncommitted root* when the clause part of the root node label is $C_?$, and said to *have a committed root* otherwise.

Example 3.1.5 $\mathcal{T}_8$ is a success tree in P. The computation tree below is a suspension tree in P.

one-by-one([0|W1],[0|Z1])
C_{09}

/ \

[0|Z1]=[0|Z1]° next-one(W1,Z1)
C_0 $C_?$

3.2 Computed Atom Behavior

In this section, we will define a more abstracted aspect of the GHC execution [21].

(1) Extension Ordering between Computation Trees

The definition of computation tree naturally introduces a partial ordering relation between computation trees. (Intuitively, this ordering means that computation tree $\mathcal{T}$ can be extended to $\mathcal{T}'$ when additional instantiation θ is applied to all the node labels of $\mathcal{T}$.)

Definition Extension Ordering between Computation Trees
Let $\mathcal{T}$ and $\mathcal{T}'$ be computation trees in program P. Then, $\mathcal{T}$ is said to be *extensible* to $\mathcal{T}'$ and denoted by $\mathcal{T} \preceq \mathcal{T}'$ when there exists a substitution θ for the variables in the root atom of $\mathcal{T}$ such that $\mathcal{T}'$ is an extension of $\mathcal{T}\theta$ in P, where $\mathcal{T}\theta$ denotes the tree obtained from $\mathcal{T}$ by applying θ to the atom part of every node label.

Example 3.2.1 $\mathcal{T}_8$ of Example 3.1.3, and $\mathcal{S}_0, \mathcal{S}_2$ of Example 3.1.4 are computation trees in P, and $\mathcal{S}_0 \preceq \mathcal{S}_2 \preceq \mathcal{T}_8$ holds.

(2) Maximal Subextension

Definition Maximal Subextension

Let $\mathcal{T}$ and $\mathcal{T}'$ be computation trees in program P such that $\mathcal{T}$ is extensible to $\mathcal{T}'$. A computation tree $\mathcal{T}''$ is called a *maximal subextension* of $\mathcal{T}$ in $\mathcal{T}'$ when

(a) $\mathcal{T}''$ is an extension of $\mathcal{T}$ in P,
(b) $\mathcal{T}''$ is extensible to $\mathcal{T}'$, and
(c) there exists no other computation tree which satisfies (a), (b) and is a proper extension of $\mathcal{T}''$ in P.

Example 3.2.2 $\mathcal{S}_2$ is the maximal subextension of $\mathcal{S}_0$ in $\mathcal{T}_8$.

(3) Computed Atom Behavior

Definition Computed Atom Behavior

Let A be an atom of the form $p(X_1, X_2, \ldots, X_n)$ where $X_1, X_2, \ldots, X_n$ are distinct variables, and $\mathcal{T}$ be a maximal computation tree in P whose root atom is $A\nu$. Then, $(A\sigma, A\tau)$ is called an *atom pair of $\mathcal{T}$ in P*, when the following conditions are satisfied:

(a) $A\sigma < A\tau \leq A\nu$.
(b) Let $\mathcal{T}_0$ be the initial tree of $A\sigma$. Then, there exists a maximal subextension of $\mathcal{T}_0$ in $\mathcal{T}$ with solution $A\tau$.
(c) There exists no other pair $(A\sigma', A\tau')$ satisfying (a),(b) and $A\sigma' < A\sigma$, $A\tau \leq A\tau'$.

The set of all the atom pairs of $\mathcal{T}$ is called a *computed atom behavior of $\mathcal{T}$*. A computed atom behavior is called a computed *success* (resp. *failure*, *suspension*) atom behavior when it is a computed atom behavior of a success (resp. failure, suspension) tree.

Example 3.2.3 The set of atom pairs

$(merge([0|X1], YY, W), merge([0|X1], YY, [0|W1])$,
$(merge([0|X1], [1|Y1], W), merge([0|X1], [1|Y1], [0, 1|W2])$,
$(merge([0|X1], [1, 1|Y2], W), merge([0|X1], [1, 1|Y2], [0, 1, 1|W3])$,
$(merge([0|X1], [1, 1], W), merge([0|X1], [1, 1], [0, 1, 1|X1])$,
$\vdots$

is the computed atom behavior of $\mathcal{T}_8$.

3.3 Computed Behavior Interpretation

Our operational semantics is as follows:

Definition Computed Behavior Interpretation

The set of all the success atom behaviors in program P is denoted by $M_1(P)$, or simply by M_1. The set of all the success atom behaviors and suspension atom behaviors in P is denoted by $M_2(P)$, or simply by M_2. The set of all the computed atom behaviors in P is denoted by $M_3(P)$, or simply by M_3. †

Example 3.3 Let P be the program of Example 2.1. Then $M(P_1)$ includes a success atom behavior of $merge([0, 0], [1, 1], [0, 1, 1, 0])$ of Example 3.2.3. $M_2(P)$ includes

$\{ (one\text{-}by\text{-}one([0|W1], Z), one\text{-}by\text{-}one([0|W1], [0|Z1])) \}$.

---

† The semantics formalized in our previous paper [14] is substantially M_1. The semantics used for the diagnosis in [21] is M_3.

4. A Fixpoint Semantics of Flat GHC Programs

4.1 Atom Behavior in General

(1) Goal Behavior

Suppose that a given goal $\Gamma\mu$ is executed together with other goals. Then, the variables in the goal might be instantiated by the execution of the goal itself. (The goal exports the instantiation to the other goals.) Or, the variables in the goal might be instantiated by the execution of the other goals. (The goal imports the instantiation from the other goals.) Suppose that the initial goal has succeeded or been suspended with the form $\Gamma\nu$ possibly after the interactions with the other goals, where we assume $\Gamma\mu < \Gamma\nu$. Then, let $\Gamma\sigma$ be a goal such that $\Gamma\mu \leq \Gamma\sigma < \Gamma\nu$, and let us execute the goal $\Gamma\sigma$ *as far as possible* separately from the other goals along the same course as the execution of $\Gamma\mu$ mentioned first. Then, since $\Gamma\sigma$ cannot import the instantiation from the other goals, it would reach a goal $\Gamma\tau$ such that $\Gamma\tau \leq \Gamma\nu$, and stop there. Then, the pair $(\Gamma\sigma, \Gamma\tau)$ denotes an interval the goal can pass autonomously. Let us collect all the intervals $(\Gamma\sigma, \Gamma\tau)$ such that $\Gamma\sigma < \Gamma\tau$ and there exists no other such interval subsuming it. Then, the set will represent some structure of the execution of $\Gamma\mu$ mentioned first.

Definition Goal Pair

A pair of goals $(\Gamma\sigma, \Gamma\tau)$ is called a *goal pair* when $\Gamma\sigma \leq \Gamma\tau$. Two goal pairs are considered identical when they are identical up to renaming of the variables appearing in the pairs.

Definition Goal Behavior

A finite set $\mathcal{B}$ of goal pairs is called a *goal behavior of* $\Gamma\nu$ when

- for any goal pair $(\Gamma\sigma, \Gamma\tau)$ in $\mathcal{B}$, there holds $\Gamma\tau \leq \Gamma\nu$, and
- $\mathcal{B}$ is marked "success," "failure" or "suspension."

A goal behavior is called a *success goal behavior* (resp. *failure goal behavior*, *suspension goal behavior*) when it is marked "success" (resp. "failure," "suspension"). Goal behaviors are denoted by $\mathcal{B}_{\Gamma\nu}$, possibly with primes and subscripts, when it is necessary to explicitly show $\Gamma\nu$, and simply by $\mathcal{B}$ when $\Gamma\nu$ is obvious from the context.

Definition Reduced Goal Behavior and Non-Reduced Goal Behavior

Let $\mathcal{B}$ be a goal behavior. A goal behavior is called a *reduced goal behavior of* $\mathcal{B}$ when it is obtained from $\mathcal{B}$ by applying the following operations as far as possible. (We will omit the uniqueness proof of the final goal behavior to be obtained.)

- Eliminate a goal pair of the form $(\Gamma\sigma, \Gamma\sigma)$.
- Replace goal pairs $(\Gamma\sigma_1, \Gamma\tau_1), (\Gamma\sigma_2, \Gamma\tau_2)$ with $(\Gamma\sigma_1, \Gamma\tau), (\Gamma\sigma_2, \Gamma\tau)$ when there exists $\Gamma\theta$ such that $\Gamma\sigma_1 \leq \Gamma\theta \leq \Gamma\tau_1$ and $\Gamma\sigma_2 \leq \Gamma\theta \leq \Gamma\tau_2$, where $\Gamma\tau$ is an m.g.u. of $\Gamma\tau_1$ and $\Gamma\tau_2$.
- Eliminate a goal pair $(\Gamma\sigma', \Gamma\tau)$ when there exists another goal pair $(\Gamma\sigma, \Gamma\tau)$ such that $\Gamma\sigma \leq \Gamma\sigma'$.

A goal behavior $\mathcal{B}$ is said to be *reduced* when $\mathcal{B}$ and $\mathcal{B}$'s reduced goal goal behavior are identical, and *non-reduced* otherwise. Hereafter, we will denote reduced goal behaviors by the term "goal behaviors" otherwise specified.

Though a goal behavior is, in general, a partially ordered set, most of the goal behaviors in this paper are totally ordered. Hereafter, success goal behaviors are depicted by arranging their component pairs lengthways (upside down w.r.t. the instantiation ordering, hence the lower a goal is located, the more it is instantiated), and by surrounding them with doubled

lines. Similarly, failure goal behaviors are surrounded with thick lines, and suspension goal behaviors with dotted lines. For example, they are depicted as below:

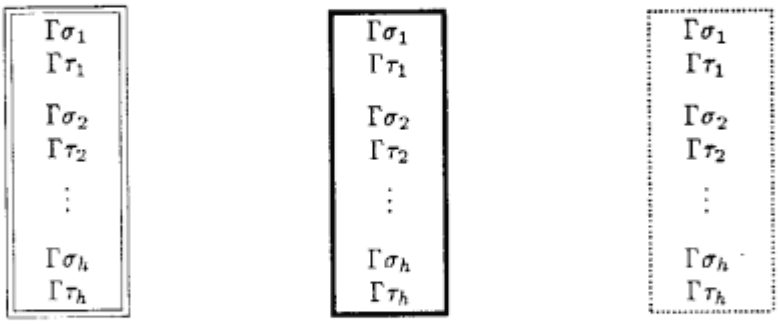


Figure 4.1 Success, Failure and Suspension Goal Behaviors

Example 4.1.1 $\mathcal{B}$ below is a success goal behavior of goal
$double(0,[0,0]), double(1,[1,1]), merge([0,0],[1,1],[0,1,1,0]), one\text{-}by\text{-}one([0,1,1,0],[0,1])$.

{double(0,XX),double(Y,YY),merge(XX,YY,W),one-by-one(W,Z)}
{double(0,[0,0]),double(Y,YY),merge([0,0],YY,[0|W1]), one-by-one([0|W1],[0|Z1])}

{double(X,[0|XX1]),double(Y,YY),merge([0|XX1],YY,W), one-by-one(W,Z)}
{double(X,[0|XX1]),double(Y,YY),merge([0|XX1],YY,[0|W1]), one-by-one([0|W1],[0|Z1])}

{double(X,XX),double(1,YY),merge(XX,YY,W),one-by-one(W,Z)}
{double(X,XX),double(1,[1,1]),merge(XX,[1,1],W), one-by-one(W,Z)}

{double(X,XX),double(Y,YY),merge(XX,YY,[0|W1]), one-by-one([0|W1],Z)}
{double(X,XX),double(Y,YY),merge(XX,YY,[0|W1]), one-by-one([0|W1],[0|Z1])}

{double(X,XX),double(Y,YY),merge(XX,YY,[0|W1]), one-by-one([0|W1],Z)}
{double(X,XX),double(Y,YY),merge(XX,YY,[0,1|W2]), one-by-one([0,1|W2],[0,1])}

$\mathcal{B}$

(2) Atom Behavior

Similar notions for atoms are defined by reducing them to those for singleton goals.

Definition Atom Pair

A pair of atoms $(A\sigma, A\tau)$ is called an *atom pair* when $(\{A\sigma\},\{A\tau\})$ is a goal pair. The goal pair $(\{A\sigma\},\{A\tau\})$ is called the *goal pair corresponding* to the atom pair $(A\sigma, A\tau)$. The ordering between two atom pairs is defined according to the ordering between the two goal pairs corresponding to the atom pairs.

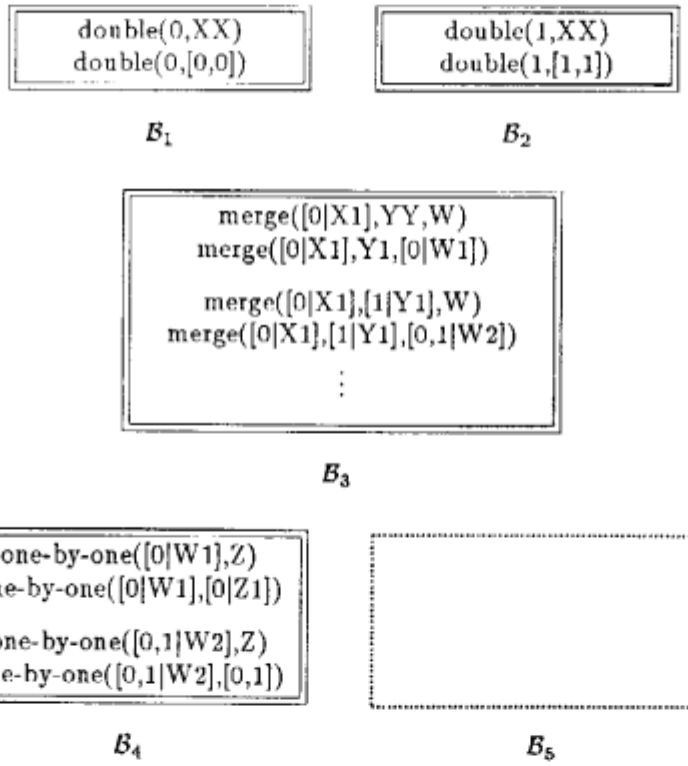
Definition Atom Behavior

A finite set $\mathcal{B}$ of atom pairs is called an *atom behavior of* $A\nu$ when the set of goal pairs obtained from $\mathcal{B}$ by replacing each atom pair with its corresponding goal pair is a goal pair of $\{A\nu\}$. A *success* (resp. *failure, suspension*) *atom behavior* is defined in the same way as a

success (resp. failure, suspension) goal pair. Atom behaviors are denoted by $\mathcal{B}, \mathcal{H}$, possibly with primes and subscripts.

Similarly to goal behaviors, success (resp. failure, suspension) goal behaviors are depicted by arranging their component pairs lengthways (upside down w.r.t. the instantiation ordering), and by surrounding them with double lines (resp. thick lines, dotted lines).

Example 4.1.2 $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ below are success atom behaviors of
$double(0,[0,0])$, $double(1,[1,1])$, $merge([0,0],[1,1],[0,1,1,0])$.
$\mathcal{B}_4$ below is a success atom behavior of
$one\text{-}by\text{-}one([0,1,1,0],[0,1])$,
while $\mathcal{B}_5$ is a suspension atom behavior of
$two\text{-}at\text{-}once([0|W1],Z)$.



$\mathcal{B}_1$ $\mathcal{B}_2$ $\mathcal{B}_3$ $\mathcal{B}_4$ $\mathcal{B}_5$

4.2 Behavior Interpretation in General

A behavior interpretation is intended to play the same role as a Herbrand interpretation.

Definition Behavior Interpretation

A *behavior interpretation* is a set of atom behaviors, and denoted by I. An atom behavior is said to be *true in* I when it is in I. Otherwise, it is said to be *false in* I.

4.3 Parallel Conjunction and Guarded Implication

To interprete the two constructs "," (in the body part) and ":- $G_1, G_2, \ldots, G_m$|" for atom behaviors and goal behaviors, we will introduce two notions, *parallel conjunction* and *guarded implication*, for them.

(1) Parallel Conjunction

Suppose that atom behaviors $\mathcal{B}_i$ of $B_i\nu$ are given for $i = 1, 2, \ldots, n$. Then, what behavior does the execution of goal $\{B_1, B_2, \ldots, B_n\}$ shows, when each atom B_i is executed along the course of computation represented by $\mathcal{B}_i$?

Definition Parallel Conjunction

Let Δ be a goal $\{B_1, B_2, \ldots, B_n\}$, $\mathcal{B}_i$ be an atom behavior of $B_i\nu$ for $i = 1, 2, \ldots, n$, and $\mathcal{B}$ be a goal behavior

$\{\ (\Delta\sigma, \Delta\tau) \mid (B_i\sigma, B_i\tau)$ is in $\mathcal{B}_i$ for some $i = 1, 2, \ldots, n\ \}$.

Then, a reduced goal behavior of $\Delta\nu$ is called the *parallel conjunction of* $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$, when it satisfy the following two conditions:

(a) It consists of all the goal pairs in $\mathcal{B}$'s reduced goal behavior.

(b) It is a success goal behavior when all $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ are success atom behaviors, a failure goal behavior when some $\mathcal{B}_i$ is a failure atom behavior, and a suspension goal behavior otherwise.

Example 4.3.1 The goal behavior $\mathcal{B}$ of Example 4.1.1 is the parallel conjunction of $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, $\mathcal{B}_4$ of Example 4.1.2.

(2) Guarded Implication

Let "H :- Γ | Δ" be a flat clause in program P, and suppose that a reduced goal behavior $\mathcal{B}$ which represents some execution of Δ is given. Then, what behavior does the execution of the head H will show if the execution is committed to this clause and the body is executed according to $\mathcal{B}$? For obtaining an atom behavior for the head H using the clause and the goal behavior $\mathcal{B}$, the following problems need to be taken into consideration:

- Even if $\Delta\sigma < \Delta\tau$, the instantiation ordering $H\sigma < H\tau$ does not necessarily hold, i.e., $H\sigma$ might be identical to $H\tau$, because H does not necessarily contain all the variables in Δ so that the substitutions for some variables in Δ are ignored.
- Even if the execution of $H\eta$ succeeds in the guard of the clause with θ, goal $\Delta\eta\theta$ is not necessarily more instantiated than or equal to $\Delta\sigma$, because σ might instantiate the variables in Δ but neither in H nor in Γ.

Definition Guarded Implication

Let C be a flat clause "H :- Γ | Δ" in program P, and $(\Delta\sigma, \Delta\tau)$ be a goal pair. Then, an atom pair $(H\eta, H\tau)$ is called the *guarded implication of* $(\Delta\sigma, \Delta\tau)$ *using* C when

(a) the execution of $H\eta$ succeeds in the guard of C with θ (without instantiating the variables in $H\eta$), where η is a substitution for the variables appearing in H,

(b) $\Delta\sigma \leq \Delta\eta\theta \leq \Delta\tau$, and

(c) there exists no other $H\eta'\theta'$ satisfying (a), (b) and more general than $H\eta\theta$.

Let $\mathcal{B}$ be a reduced goal behavior of $\Delta\nu$, and $\mathcal{H}$ be the set of all the guarded implication of goal pairs in $\mathcal{B}$ using C. Then, $\mathcal{H}$'s reduced atom behavior (of $H\nu$) is called the *guarded implication of* $\mathcal{B}$ *using* C.

Note that, due to the restriction on the primitive predicates and flat clauses mentioned in Section 2, the guarded implication of a goal pair (if exists) is unique up to renaming of variables. (See Appendix.)

Example 4.3.2 Recall the clause C_{01} of Example 2.1 as below:

```
p1(X,Y,Z) :- |
          double(X,XX), double(Y,YY), merge(XX,YY,W), one-by-one(W,Z).
```

Then, atom behavior

$\{(p1(0,Y,Z), p1(0,Y,[0|Z1]), (p1(0,1,Z), p1(0,1,[0,1])\}$

is the guarded implication of $\mathcal{B}$ of Example 4.1.1 using C_{01}.

4.4 Transformation of Behavior Interpretations

Using the two notions defined in the previous section, we will introduce three transformations, T_1, T_2 and T_3.

Definition Basic Transformation

The basic transformation T_0 of behavior interpretations associated with program P is defined as below:

$T_0(I) = \{\mathcal{H} |$ P contains a clause C of the form "H :- $G_1, G_2, \ldots, G_m \mid B_1, B_2, \ldots, B_n$," I contains atom behaviors $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ of $B_1\nu, B_2\nu, \ldots, B_n\nu$, respectively, $\mathcal{H}$ is the guarded implication of the parallel conjunction of $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ using C $\}$.

Definition Base Behavior Interpretation for Immediate Success, Suspension and Failure

Let $\mathcal{B}_{t=t}$ be the set of all the equation pairs $(s_1 = s_2, s = s)$ such that

(a) s_1 and s_2 are unifiable, but different terms,

(b) s is a most general unification of them, and is more general than or equal to t, and

(c) there exists no other equations $(s_1' = s_2', s = s)$ satisfying (a),(b) and more general than $(s_1 = s_2, s = s)$,

and let $\mathcal{B}_{t_1=t_2}$ be an empty atom behavior of "$t_1 = t_2$" for different terms t_1, t_2. Then, the behavior interpretations I_1, I_2 and I_3 defined by

$I_1 = \{\mathcal{B}_{t=t} |$ t is a term $\}$,

$I_2 = \{\{\ \}_A |$ A is suspended immediately in P $\}$,

$I_3 = \{\mathcal{B}_{t_1=t_2} |$ t_1 and t_2 are non-unifiable terms $\}$

are called the *base behavior interpretation for immediate success*, *suspension*, and *failure*, respectively.

Definition Transformations of Behavior Interpretations

Let T_0 be the basic transformation associated with program P, and I_1, I_2, I_3 be the base bahavior interpretations for immediate success, suspension and failure, respectively. Then, the transformations T_1, T_2, T_3 of behaviors interpretations defined by

$T_1(I) = I_1 \cup T_0(I)$,

$T_2(I) = I_1 \cup I_2 \cup T_0(I)$,

$T_3(I) = I_1 \cup I_2 \cup I_3 \cup T_0(I)$

are called the *transformations associated with P*.

4.5 Least Fixpoint of the Transformation

The three transformations T_1, T_2, T_3 of behavior interpretations are obviously monotonic w.r.t. the set inclusion ordering of behavior interpretations, i.e.,

if $I \subseteq J$ then $T_1(I) \subseteq T_1(J)$,
if $I \subseteq J$ then $T_2(I) \subseteq T_2(J)$,
if $I \subseteq J$ then $T_3(I) \subseteq T_3(J)$.

Hence, from Knaster-Tarski's fixpoint theorem, there exist the least fixpoints of T_1, T_2, T_3.

Definition Least Fixpoint of T_1, T_2, T_3

Let T_1, T_2, T_3 be the transformations of behavior interpretations associated with program P. Then, the least fixpoints of T_1, T_2, T_3 are denoted by $lfp(T_1), lfp(T_2), lfp(T_3)$, respectively.

5. Equivalence of the Operational Semantics and the Fixpoint Semantics

We have introduced an operational semantics in Section 3, and a fixpoint semantics in Section 4. In this section, we will show that those two semantics are equivalent.

Theorem 5 Equivalence of the Operational Semantics and the Fixpoint Semantics

Let P be a flat GHC program. Then, $M_1 = lfp(T_1)$, $M_2 = lfp(T_2)$ and $M_3 = lfp(T_3)$.

Proof. Due to space limit, we will show only the proof of the first equivalence. (Other two cases are proved in the same way.) The proof is divided into two parts, the "Left Inclusion" part and the "Right Inclusion" part.

Left Inclusion: $M_1 \subseteq lfp(T_1)$.

Let $\mathcal{B}$ be an atom behavior in M_1 and $\mathcal{T}$ be the corresponding computation tree. Then, the inclusion relation is proved easily by induction on the structure of computation trees. (This proof implies $M_1 = \bigcup_{k=0}^{\infty} T_1^k(\emptyset)$.)

Right Inclusion: $M_1 \supseteq lfp(T_1)$.

We will prove the continuity of T_1 directly, i.e., for any chain of behavior interpretations

$$J_1 \subseteq J_2 \subseteq \cdots \subseteq J_k \subseteq \cdots,$$

there holds

$$T_1(\bigcup_{k=0}^{\infty} J_k) = \bigcup_{k=0}^{\infty} T_1(J_k).$$

One direction $T_1(\bigcup_{k=0}^{\infty} J_k) \supseteq \bigcup_{k=0}^{\infty} T_1(J_k)$ is easy to prove. The other direction $T_1(\bigcup_{k=0}^{\infty} J_k) \subseteq \bigcup_{k=0}^{\infty} T_1(J_k)$ is proved as follows: Suppose that $\mathcal{H}$ is in $T_1(\bigcup_{k=0}^{\infty} J_k)$. Then, from the definition of T_1,

(a) P contains a clause C of the form "H :- $\Gamma | B_1, B_2, \ldots, B_n$,"
(b) $\bigcup_{k=0}^{\infty} J_k$ containsn $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$, and
(c) $\mathcal{H}$ is the guarded implication of the parallel conjunction of $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ using C.

Then, for some i,

(b)' $\bigcup_{k=0}^{i} J_k$ containsn $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$,

which implies that $\mathcal{H}$ is in $\bigcup_{k=0}^{i} T^k(J_k)$, hence, $\mathcal{H}$ is in $\bigcup_{k=0}^{\infty} T^k(J_k)$. (Note that, since we have not yet fully developed the model theoretical semantics e.g., the relation between the models and T, the model intersection property, etc., we cannot proceed along the same line as van Emden and Kowalski [5], in which the continuity was not referred directly.)

6. An Example — Brock-Ackerman's Anomaly —

Recall the flat GHC program of Example 2.1.

C_{01}:
```
p1(X,Y,Z) :- |
        double(X,XX), double(Y,YY), merge(XX,YY,W), one-by-one(W,Z).
```

```
C02: p2(X,Y,Z) :- |
         double(X,XX), double(Y,YY), merge(XX,YY,W), two-at-once(W,Z).
C03: double(0,AA) :- | AA=[0,0].
C04: double(1,AA) :- | AA=[1,1].
C05: merge([A|Xs],Ys,Zs) :- | Zs=[A|Zs1], merge(Xs,Ys,Zs1).
C06: merge(Xs,[A|Ys],Zs) :- | Zs=[A|Zs1], merge(Xs,Ys,Zs1).
C07: merge([ ],Ys,Zs) :- | Zs=Ys.
C08: merge(Xs,[ ],Zs) :- | Zs=Xs.
C09: one-by-one([A|W],Z) :- | Z=[A|Z1], next-one(W,Z1).
C10: next-one([B|W],Z) :- | Z=[B].
C11: two-at-once([A,B|W],Z) :- | Z=[A|Z1], Z1=[B].
C12: complement([0|Z],Y) :- | Y=1.
C13: complement([1|Z],Y) :- | Y=0.
```

Then, the atom behavior of "$p1(0,1,[0,1])$" in $M_1(P)$ is a set consisting of two atom pairs

$\{ (p1(0,Y,Z), p1(0,Y,[0|Z1])), (p1(0,1,Z), p1(0,1,[0,1])) \}$,

while that of "$p2(0,1,[0,1])$" in $M_1(P)$ is a set consisting of only one atom pair

$\{ (p2(0,1,Z), p2(0,1,[0,1])) \}$.

Hence, the conjunction of atom behaviors of "$p1(0,1,[0,1])$" and "$complement([0,1],1)$" includes a goal pair

$(\{p1(0,Y,Z), complement(Z,Y)\}, \{p1(0,1,[0,1]), complement([0,1],1)\})$,

while that of "$p2(0,1,[0,1])$" and "$complement([0,1],1)$" does not.

7. Comparison with the Semantics of Prolog Programs

Suppose that we identify any Prolog clause of the form

"$p(t_1,t_2,\ldots,t_m)$:- $B_1,B_2,\ldots,B_n$."

with a GHC clause of the form

"$p(X_1,X_2,\ldots,X_m)$:- $\mid X_1=t_1,X_2=t_2,\ldots,X_m=t_m,B_1,B_2,\ldots,B_n$."

and compare a Prolog program with the GHC program consisting of such converted clauses.

As for the original Prolog program, the semantics of the program is usually defined by

$M(P) = \{A\nu | A\nu$ is a ground atom which succeeds in $P\}$

or, a little more generally, by

$M_{ext}(P) = \{A\nu | A\nu$ is an atom which succeeds in $P\}$

Intuitively, this is the set of all ground (or not necessarily ground) atoms that is provable from program P [1],[5],[19].

As for the converted GHC program, as far as the predicate of an atom is not undefined, the execution of the atom in such a GHC program is never suspended immediately, since there always exists some clause to which the execution is committable. Therfore, the following holds for such a GHC program:

- A success atom behavior always consists of a single atom pair of the form
  $(p(X_1,X_2,\ldots,X_n), p(t_1,t_2,\ldots,t_n))$,
  where $X_1,X_2,\ldots,X_n$ are distinct variables. (Hence, the first element of the pair gives no information to us.)
- A suspension atom behavior is always for the atom with an undefined predicate, and is an empty set. (Usually, undefined predicates are of no concern to us.)

Hence, let us define the semantics of the program as below:

$\tilde{M}(P) = \{(A\tau, A\nu) | \{(A\sigma, A\tau)\}$ is a success atom behavior of ground atom $A\nu$ in $P\}$.
or, a little more generally, by

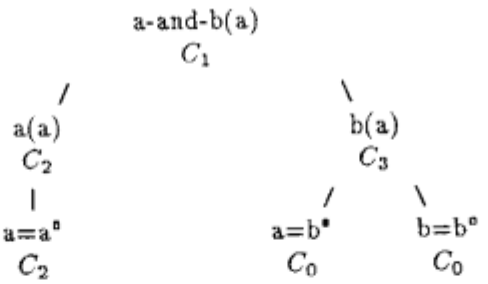
$\tilde{M}_{ext}(P) = \{(A\tau, A\nu) | \{(A\sigma, A\tau)\}$ is a success atom behavior of atom $A\nu$ in $P\}$.
Intuitively, this is the set of all atom pairs $(A\tau, A\nu)$ such that the ground (or not necessarily ground) atom $A\nu$ is provable from P, and it is provable in the same way even if it is as general as (universally quantified atom) $A\tau$. (Cf. [6],[10].)

8. Discussion — Truly Parallel v.s. Non-deterministic Sequential —

Recall that we have formalized the notion of computation tree based on the non-deterministic sequential GHC execution, in which the instantiation caused by the extension at one node is propagated immediately to all the other nodes. (Let us call such execution *non-deterministic sequential*.) At first glance, it seems unnatural, since we cannot guarantee that the instantiation is propagated in such a way. If nodes are assigned to different processors, the extension at some node might be done before the instantiation caused at other nodes has been propagated. (Let us call such execution *truly parallel*.) For example, consider the following program:

C_1: `a-and-b(X) :- | a(X), b(X).`
C_2: `a(X) :- | X=a.`
C_3: `b(X) :- | X=b, X=b.`

Suppose that the substitution $< X \Leftarrow a >$ caused by the execution of "$X = a$" is propagated before the extension at the left node labelled with "$X = b$," but after the extension at the right node labelled with "$X = b$". Then, we will have the tree below, which is not our maximal computation tree.



However, note that, if we can assume that the instantiation caused at each node is eventually propagated to all the other nodes, we can say that

- a labelled tree is a success tree by non-deterministic sequential execution if and only if it is a success tree by truly parallel execution, and
- a labelled tree is a suspension tree by non-deterministic sequential execution if and only if it is a suspension tree by truly parallel execution.

Hence, the notions of success tree and suspension tree do not depend on the non-deterministic sequential execution mechanism we have employed in Section 3.1. Moreover, once a maximal computation tree (by non-deterministic sequential execution) is given, any maximal subextension in it, hence its atom behavior, is also independent of the sequentiality.

9. Conclusions

We have presented a fixpoint semantics of flat GHC programs. Further refinement and logical formalization of the semantics are left for future.

Acknowledgements

This research was done as a part of the Fifth Generation Computer Systems project of Japan. We would like to thank Dr. K. Fuchi (Director of ICOT) for the opportunity of doing this research, and Dr. K. Furukawa (Deputy Director of ICOT), Dr. R. Hasegawa (Chief of ICOT 1st Laboratory) and Dr. H. Ito (Former Chief of ICOT 3rd Laboratory) for their advice and encouragement.

References

[1] Apt, K.R. and M.H.van Emden, "Contribution to the Theory of Logic Programming," J. ACM, Vol.29, No.3, pp.841–862, 1982.

[2] Beckman, L., "Towards a Formal Semantics for Concurrent Logic Programming Languages," Proc. of 3rd International Conference on Logic Programming, pp.335–349, London, July 1986.

[3] Bowen, D.L, L.Byrd, F.C.N.Pereira, L.M.Pereira and D.H.D.Warren, "DECsystem-10 Prolog User's Manual," Department of Artificial Intelligence, University of Edinburgh, 1983.

[4] Brock, J.D. and W.B.Ackerman, "Scenario : A Model of Nondeterminate Computation," in *Formalization of Programming Concepts* (J.Diaz and I.Ramos Eds.), Lecture Notes in Computer Science 107, pp.252–259, Springer-Verlag, 1981.

[5] van Emden, M. and R.Kowalski, "The Semantics of Predicate Logic as a Programming Language," J. of ACM, Vol.23, No.4, pp.733–742, 1976.

[6] Falaschi, M., G.Levi, M.Martelli and C.Palamidessi, "A More General Declarative Semantics for Logic Programming Languages," to appear, 1988.

[7] Fujita, H., A.Okumura and K.Furukawa, "Partial Evaluation of GHC Programs Based on UR-set with Constraint Solving," Proc. of Logic Programming '88, Seattle, August 1988.

[8] Furukawa, K. and K.Ueda, "GHC Process Fusion by Program Transformation," Proc. of 2nd Conference of Japan Society for Software Science and Technology, pp.89–92, Tokyo, November 1985.

[9] Furukawa, K., A.Okumura and M.Murakami, "Unfolding Rules for GHC Programs," Proc. of 2nd France-Japan Artificial Intelligence and Computer Science Symposium, Cannes, November 1987. Also Proc. of 4th Conference of Japan Society for Software Science and Technology pp.267–270, Kyoto, November 1987.

[10] Gaifman, H. and E.Shapiro, "A Fully Abstract Compositional Semantics for Logic Programs," Conf. Rec. of 16th Annual ACM Symposium on Principles of Programming Languages, pp.134–142, Austin, Texas, January 1989.

[11] Gerth, R., M.Codish, Y.Lichtenstein and E.Shapiro, "Fully Abstract Denotational Semantics for Flat Concurrent Prolog," Proc. of 3rd IEEE Annual Symposium on Logic in Computer Science, pp.320–333, Edinburgh, 1988.

[12] Huntbach, M., "Algorithmic Parlog Debugging," Proc. 4th Symposium on Logic Programming, pp.288–297, San Francisco, August 1987.

[13] Kameyama, Y., "Axiomatic System for Concurrent Logic Programming Languages," Master's Thesis of The University of Tokyo, 1987. Also an extended abstract was presented at U.S.-Japan Workshop on Logic of Programs, Honolulu, May 1987.

[14] Kanamori,T. and M.Maeji, "A Preliminary Note on the Semantics of GHC Programs," ICOT Technical Report TR-434, ICOT, Tokyo, December 1988.

[15] Levi, G. and C.Palamidessi, "An Approach to the Declarative Semantics of Synchronization in Logic Language," Proc. of 4th International Conference on Logic Programming, pp.877–893, Melbourne, May 1987.

[16] Levi, G., "A New Declarative Semantics of Flat Guarded Horn Clauses," to appear, January 1988.

[17] Lichtenstein, Y. and E.Shapiro, "Concurrent Algorithmic Debugging," The Weizmann Institute of Science, Department of Computer Science, Technical Note CS87-20, 1987.

[18] Lichtenstein, Y. and E.Shapiro, "Abstract Algorithmic Debugging," Proc. of 1987 Symposium on Logic Programming, pp. 512–531, San Francisco, August 1987.

[19] Lloyd, J.W., "Foundations of Logic Programming," Springer-Verlag, 1984.

[20] Lloyd, J.W. and A.Takeuchi, "A Framework of Debugging GHC Programs," ICOT Technical Report TR-186, ICOT, Tokyo, 1986.

[21] Maeji,M. and T.Kanamori, "GHC Program Diagnosis Using Atom Behavior," ICOT Technical Report TR-5??, ICOT, Tokyo, March 1990.

[22] Maher, M.J., "Logic Semantics for A Class of Committed-choice Programs," Proc. of 4th International Conference on Logic Programming, pp.858–876, Melbourne, May 1987.

[23] Murakami, M., "Proving Partial Correctness of Guarded Horn Clauses," Proc. of The Logic Programming Conference '87, pp.117–124, Tokyo, June 1987. Also to appear in *Logic Programming '87* (E.Wada Ed.),1988.

[24] Murakami, M., "On an Axiomatic Semantics of GHC Programs," (in Japanese) Proc. of 4th Conference of Japan Society for Software Science and Technology, pp.259–262, Kyoto, November 1987.

[25] Murakami, M., "A Declarative Semantics of Parallel Logic Programs with Perpetual Processes," to appear Proc. of International Conference on Fifth Generation Computer Systems 1988, Tokyo, November 1988.

[26] Saraswat, V.A., "Partial Correctness Semantics for CP(!,–)," Proc. of the Foundation of Software Technology and Theoretical Computer Sciences Conference, Lecture Notes in Computer Science 206, pp.347–368, Springer-Verlag, 1985.

[27] Saraswat, V.A., GHC : Operational Semantics, Problems and Relationship with CP(|, ,|)," Proc. of 1987 Symposium on Logic Programming, pp.347–358, San Francisco, August 1987.

[28] Shibayama, E., "A Compositional Semantics of GHC," Proc. of 4th Conference of Japan Society for Software Science and Technology, pp.255–258, Kyoto, November 1987.

[29] Takeuchi, A., "Algorithmic Debugging of GHC Programs," ICOT Technical Report TR-185, ICOT, Tokyo, 1986.

[30] Takeuchi, A., "Towards A Semantic Model of GHC," Technical Report of "Foundation of Software" Research Group, Information Processing Society of Japan, 1986.

[31] Ueda, K., "Guarded Horn Clauses," ICOT Technical Report TR-103, ICOT, Tokyo, 1985. Also in Proc. of The Logic Programming Conference '85, pp.225–236, 1985. Also in *Logic Programming '85* (E.Wada Ed.), Lecture Note in Computer Science 221, pp.168–179, Springer-Verlag, 1986. Also in *Concurrent Prolog: Collected Papers* (E.Y.Shapiro Ed.), Vol. 1, Chap. 4, The MIT Press, 1987.

[32] Ueda, K., "On the Operational Semantics of Guarded Horn Clauses," Presented at RIMS Simposium on Mathematical Methods in Software Science and Engineering '85, Kyoto, October 1985. Also RIMS Research Report 586, pp.263–283, Research Institute for Mathematical Sciences, Kyoto University, March 1986.

[33] Ueda, K., "Guarded Horn Clauses," Doctorial Thesis, Information Engineering Course, Faculty of Engineering, University of Tokyo, 1986.

[34] Ueda, K., "Guarded Horn Clauses : A Parallel Logic Programming Language with the Concept of a Guard," Proc. of 1st France-Japan Artificial Intelligence and Computer Science Symposium, pp.127–138, Tokyo, October 1987. Also in *Programming of Future Generation Computers* (M.Nivat and K.Fuchi Eds.), North-Holland, 1988.

[35] Ueda, K. and K.Furukawa, "Transformation Rules for GHC Programs," to appear as ICOT Technical Report, ICOT, Tokyo, 1988. Also to appear Proc. of International Conference on Fifth Generation Computer Systems 1988, Tokyo, November 1988.

Appendix. The Well-definedness of Guarded Implication

In Section 4.3, we have defined the guarded implication as follows:

Definition Guarded Implication

Let C be a flat clause "H :- $\Gamma \mid \Delta$" in program P, and $(\Delta\sigma, \Delta\tau)$ be a goal pair. Then, an atom pair $(H\eta, H\tau)$ is called the *guarded implication of* $(\Delta\sigma, \Delta\tau)$ *using* C when

(a) the execution of $H\eta$ succeeds in the guard of C with θ (without instantiating the variables in $H\eta$), where η is a substitution for the variables appearing in H,

(b) $\Delta\sigma \leq \Delta\eta\theta \leq \Delta\tau$, and

(c) there exists no other $H\eta'\theta'$ satisfying (a), (b) and more general than $H\eta\theta$.

Let $\mathcal{B}$ be a reduced goal behavior of $\Delta\nu$, and $\mathcal{H}$ be the set of all the guarded implication of goal pairs in $\mathcal{B}$ using C. Then, $\mathcal{H}$'s reduced atom behavior (of $H\nu$) is called the *guarded implication of* $\mathcal{B}$ *using* C.

Then, under what conditions does the guarded implication of a goal pair using a clause exist, and is it unique if it exists?

First of all, if the guarded implication of $(\Delta\sigma, \Delta\tau)$ using C exists, the execution of $H\tau$ must be committable to clause C, hence

(A) $H\tau$ must succeed in the guard of C.

Since the guard atoms are all primitive atoms, when $H\tau$ succeeds in the guard of C, the execution of an instance of each guard atom $p(s_1, s_2, \ldots, s_n)$ of C is committed to some clause of the form

$p(t_1, t_2, \ldots, t_n)$:- | E,

where E is a set of equations. Let $\mathcal{E}$ be the union of the set of all the equations

$\{s_1 = t_1, s_2 = t_2, \ldots, s_n = t_n\} \cup E$

w.r.t. all the guard atoms of C, and η_0 be the most general unifier of $\mathcal{E}$. Then, $H\eta_0$ can succeed in the guard of C in the same way as $H\tau$, and moreover, is most general among such atoms.

Definition Most General Atom for Commitment

Let C be a flat clause of the form

H :- G_1, G_2, ..., G_m | B_1, B_2, ..., B_n,

$H\tau$ be an atom such that the execution of $H\tau$ succeeds in the guard of C. Then, an atom $H\eta_0$ is called the *most general atom for commitment* when

(a) $H\eta_0$ succeeds in the guard of C in the same way as $H\tau$, and

(b) there exists no other $H\eta_0'$ satisfying (a) and more general than $H\eta_0$.

Suppose that the most general atom $H\eta_0$ for commitment succeeds with substitution θ_0. (The domains of η_0 and θ_0 are disjoint.) Let η and θ be the substitutions in the definition of the guarded implication of $(\Delta\sigma, \Delta\tau)$. Then,

(a) $\Delta\eta_0\theta_0 \leq \Delta\eta\theta$,
(b) $\Delta\sigma \leq \Delta\eta\theta \leq \Delta\tau$, and
(c) there exists no other $H\eta'\theta'$ satisfying (a) and (b) and more general than $H\eta\theta$.

Then, let $\mathcal{V}$ be the set of all the variables appearing in Δ but not in the guard part. Since $\eta\theta$ has no relation with the variables in $\mathcal{V}$, σ must not have a relation with the variables in $\mathcal{V}$ from (b) above, hence

(B) The variables in $\mathcal{V}$ must not appear either in the domain or the range of σ.

Suppose that σ satisfies the condition above. Then, $\Delta\eta\theta$ is an m.g.u. of $\Delta\sigma$ and $\Delta\eta_0\theta_0$ from (a),(b) and (c) above, hence,

(C) η is the restriction of an m.g.u. of σ and $\eta_0\theta_0$ to the variables in H.

Then, it is obvious that the three conditions (A),(B),(C) are sufficient for $H\eta$ to satisfy the definition of the guarded implication of $(\Delta\sigma, \Delta\tau)$. Hence, the guarded implication of a goal pair is unique up to renaming of variables when it exists.