ICOT Technical Report: TR-614

TR-614

A Concurrent Operational Semantics of Flat GHC Programs

by

M. Murakami (Fujitsu)

January, 1991

A Concurrent Operational Semantics of Flat GHC Programs

Extended Abstract

Masaki Murakami

International Institute for Advanced Study of Social Information Science,
Fujitsu, 17-25, Shinkamata 1-chome, Ota-ku, Tokyo 144, Japan
e-mail: murakami@iias.fujitsu.co.jp

1 Introduction

In this paper, an operational semantics of GHC is presented. The semantics presented here is not based on a sequential state-transition model but on a concurrent top-down derivation procedure and uses the partial order model that is similar to the declarative semantics [Murakami 88] of GHC. Thus the semantics of a goal is defined with a set of guarded streams. A guarded stream is a partially ordered set of guarded unifications and represents a reactive computation of the goal. The procedure proceeds the computation of a goal clause by building a tuple of computation trees such that each node is labeled by a tuple of (sub)goal and a guarded stream. The label of the root node of each tree denotes the computation of each atom in the goal clause.

The declarative/fixpoint semantics [Murakami 88] is designed as the basis to show the soundness of the unfolding transformation rules and of verification rules . Thus it is designed that the semantics of a goal clause is defined in compositional manner from the semantics of each goal. There is no immediate correspondence with the informal computation rule of GHC programs.

The operational semantic presented in this paper is designed to fill the gap of the informal operational semantics and the declarative/fixpoint semantics.

2 Communication Channel

The semantics presented here is a top down derivation procedure that build a model of an infinite computation of a GHC program. The procedure is represented in a parallel imperative language. We denotes the parallel composition of process $P_1, \ldots, P_n$ as:

$$\textbf{parabegin} \quad P_1 // \ldots // P_n \ \textbf{paraend}.$$

The communications and synchronization are performed with the feature that is an extension of stream communication.

That proceeds the computations of each goal in a goal clause in parallel and produces the guarded streams for each goal in parallel by subprocesses. Thus the subprocess that process each goal cannot output an infinite partially ordered set as the whole result of the computation of the goal. In the procedure such that build an infinite structure, each subprocess should output each element of the partially ordered set during the computation. Usually, such kind of communication is stream based communication. Streams are linearly ordered sets. The data elements are produced sequentially in the producer process. The communications using streams are controlled with the linear order.

However the derivation procedure presented here consists of processes which produce partially ordered sets. The data elements of are produced in parallel in the producer process. The communications are controlled with the partial order. Thus we extend the notion of streams to partial order structures, that are called *communication channels*. Namely, the communication channel defined here is a set type variable that read/write operations are defined which are controlled with the partial order on the set.

The producer P of the communication channel S generates elements successively and adds to S. P can add only elements that are not smaller than any elements which are already in S. The consumer C of S can read only elements that are minimal elements of the unread part of S.

The producer of each communication channel is unique for each channel but more than one consumers can read the channel. The communication channel S is an output variable for the producer of S.

The operation that add an element A to output channel S is denoted as:

$$\mathbf{put}(A, S).$$

This operation can be executed when A is not smaller than any elements already in S and the result of the operation is $\{A\} \cup |S|$ where $|S|$ denotes the set of elements which are in S. If there is an element in S that is larger that A, then this operation fails.

On the other hand, the operation that read an element to X from input channel S is denoted as follows.

$$\mathbf{get}(S, X).$$

If there are elements that are unread, then a minimal element a is selected from the unread part non-deterministically, and let a be the content of X. We assume that there is a fairness of the selection of elements from the unread part of S. Namely, any elements added to S will be eventually read unless the consumer terminates. If there is no unread element in S then $\mathbf{get}(S, X)$ suspends. Note that the unread part of S is locally defined to each consumer. Thus any execution of input operation in a consumer process does not affect to any other consumers.

3 Derivation Procedure

In this section, the derivation procedure is presented. In the case of GHC programs, the execution of a goal clause does not start with whole input but input bindings are given from the environment during the execution. That situation is represented with the notion of *constraint net* in this paper. A constraint net is a set $T = \{\sigma_1, \sigma_2, \ldots\}$ where each σ_i is a finite set of unification goals, and holds the following conditions.

1) For any finite subset T' of T, if T' is closed form below, then $P = \bigcup_{\sigma \in T'} \sigma$ defines a substitution by executing all unification goals in P. P is called a *prefix* of T.

2) For any σ' and any σ_j if $\sigma_j \subset \sigma_i$ then there is a unification goal $U \in \sigma_i$ such that $\sigma_j \not\models U$.

A constraint net is a partially ordered with set inclusion.

In the rest of this paper, the reduction procedure is presented. The procedure takes a program D, a goal clause $[G_1, \ldots, G_n]$ and a constraint net T_0 as the inputs, and produces a guarded stream GU_0 that is a trace representing one of the computations of $[G_1, \ldots, G_n]$ on D with input T_0. A guarded stream is a set of guarded unifications. A guarded unification is a tuple $< \sigma|U >$ where σ is a finite set of unification goals that representing input constraint which should be solved to execute the unification goal U. σ is called *guard part* of $< \sigma|U >$ and U is called body part. A guarded stream GU is partially ordered such that for $< \sigma_1|U_1 >, < \sigma_2|U_2 > \in GU$, if $\sigma_1 \subset \sigma_2$ then $< \sigma_1|U_1 > \leq < \sigma_2|U_2 >$
The top level procedure : RUN_GOAL_CLAUSE is as follows.

```
procedure:
$RUN\_GOAL\_CLAUSE$ (input:$[G_1, \ldots G_n], D, T_0$, output: $GU_0$) :
begin
parbegin
$RUN\_GOAL(G_1, D, T_1, GU_1)$//
$MAKE\_TRANSACTION([GU_2, \ldots, GU_n], T_0, T_1)$//
$RUN\_GOAL(G_2, D, T_2, GU_2)$//
$MAKE\_TRANSACTION([GU_1, GU_3, \ldots, GU_n], T_0, T_2)$//

                                        ...

$RUN\_GOAL(G_i, D, T_i, GU_i)$//
$MAKE\_TRANSACTION([GU_1, \ldots, GU_{i-1}, GU_{i+1}, GU_n], T_0, T_i)$//

                                        ...

$RUN\_GOAL(G_n, D.T_n, GU_n)$//
$MAKE\_TRANSACTION([GU_1, \ldots, GU_{n-1}], T_0, T_n))$//
```

```
$MERGE\_TRACE([GU_1, \ldots, GU_n], GU_0)$
parend
end
```

The procedure $MAKE\_TRANSACTIONS$ is a procedure that produces a input constraint net for each goal G_i from T_0 and the output of the neighborhood goals.

```
procedure:
$MAKE\_TRANSACTION$ (input:$[GU_1, \ldots, GU_{i-1}, GU_{i+1}, GU_n], T_0$, output: $T_i$)
parbegin
repeat
    get($X, T_0$);
    put($X, T_i$)
end_repeat //
repeat
    get($GU_1, gu$);
    $U :=$ the body part of $gu$;
    put($U, T_i$)
end_repeat//
                                        ...
repeat
    get($GU_{i-1}, gu$);
    $U :=$ the body part of $gu$;
    put($U, T_i$)
end_repeat//
repeat
    get($GU_{i+1}, gu$);
    $U :=$ the body part of $gu$;
    put($U, T_i$)
    end_repeat//
                                        ...
repeat
    get($GU_n, gu$);
    $U := gu$ the body part of $gu$;
    put($U, T_i$)
end_repeat
parend
end
```

The procedure $MERGE\_TRACE$ is a merge procedure to make the output GU_0 from $GU_1, \ldots, GU_n$.

```
procedure:
$MERGE\_TRACE$(input: $GU_1, \ldots, GU_n$, output: $GU_0$)
parbegin
    repeat
        get($GU_1, X_1$);
       $\sigma_1 :=$ the guard part of $X_1$;
       $U_1 :=$ the body part of $X_1$;
          for all $U \in \sigma_1$
           begin
            if $\exists GU' \subset GU_0 \wedge |GU'|$ is a minimal model of $U$;
            then
            $G_1 := \bigcup_{gu \in GU'} \{u | u \in$ is an element of guard part of $gu\}$
            $\sigma'_1 := \sigma_1 - \{U\} \cup G_1$;
           end;
        put($< \sigma'_1|U_1 >, GU_0$)
 end_repeat
    //
                                        ...
    //
 repeat
        get($GU_n, X_n$);
       $\sigma_n$ := the guard part of $X_n$;
       $U_n$ := the body part of $X_n$;
    for all $U \in \sigma_n$
          begin
           if $\exists GU' \subset GU_0 \wedge |GU'|$ is a minimal model of $U$;
           then
           $G_n := \bigcup_{gu \subset GU'} \{u|u \in$ is an element of guard part of $gu\}$
           $\sigma'_n := \sigma_n - \{U\} \cup G_n$;
          end;
       put($< \sigma'_n|U_n >, GU_0$)
end_repeat
parend
end
```

RUN_GOAL is a procedure that takes a goal G, a program D and a constraint net T and output the guarded stream GU that denotes a computation of G on D with T.

```
procedure:
RUN_GOAL(input: $G, D, T$, output: $GU$)
$S_0 := \phi$;
repeat
    get($T, \sigma$);
    $S_0 := \sigma \cup S_0$;
until
    begin
       There exists a renaming of:

                    $C: H \text{:-} U_{g1}, \ldots, U_{gm}|U_{b1}, \ldots, U_{bh}, B_1, B_2, \ldots, B_n$

       in $D$ such that:
        (1) There exists a substitution $\sigma$ such that $G = \sigma_0 H$ .
        (2) For all $U_{gi}$, $\sigma_0 \cup S_0 \models U_{gi}$ .
    end;
parbegin
if $S_0 \cup |GU| \models U_{b1}$ then
    put($< \{U_{g1}, \ldots, U_{gm}\}|U_{b1}? >, GU$)
    else if $U_{b1}$ is consistent with $S_0 \cup |GU|$
       then
       put($< \{U_{g1}, \ldots, U_{gm}\}|U_{b1} >, GU$)
       else  fail
//
                                        ...
//
if $S_0 \cup |GU| \models U_{bh}$ then
      put($< \{U_{g1}, \ldots, U_{gm}\}|U_{bh}? >, GU$)
   else if $U_{b1}$ is consistent with $S_0 \cup |GU|$
      then
        put($< \{U_{g1}, \ldots, U_{gm}\}|U_{bh} >, GU$)
        else  fail
//
repeat
    get($T, \sigma$);
    put($\sigma, T_{body}$)
end_repeat
//
RUN_GOAL_CLAUSE
        ($[(\{U_{g1}, \ldots, U_{gm}\} \cup S_0)B_1, \ldots, (\{U_{g1}, \ldots, U_{gm}\} \cup S_0)B_n], D, T_{body}, GU_{body}$)
//
UPDATE_STREAM($GU_{body}, C, GU$)
parend
end
```

UPDATE_STREAM is the procedure that takes the results of the computations of $B_1, \ldots, B_n$ as inputs then updates GU.

```
procedure:
UPDATE_STREAM(input: $GU_{body}$ output: $GU$)

repeat
    get($GU_{body}, < \sigma | U_b >$)
    $\sigma' := \sigma \cup \{U_{g1}, \ldots, U_{gm}\} \setminus \{U_{b1}, \ldots, U_{bh}\}$;
    $gu := < \sigma' | U_b >$;
    put($gu, GU$)
end_repeat
end
```

4 Conclusion

In this paper, an operational semantics of GHC is presented as an parallel reduction procedure. This semantics is free from the discussion about fairness of scheduling of processes and Atomicity of actions.

Acknowledgments
This work has been carried out as a part of the Fifth Generation Computer Project of Japan. I would like to thank the members of The Parallel Meta System Group of IIAS-SIS for their helpful discussions.

References

[Murakami 88] M. Murakami, A New Declarative Semantics of Parallel Logic Programs with Perpetual Processes, Proc. of Int. Conf.on Fifth Generation Computer System 1988, (1988) to appear in Theoret. Comp .Sci.

[Saraswat 89] V. Saraswat, Concurrent Constraint Programming Languages , Ph. D thesis, Carnegie-Mellon University, Computer Science Department. (1989)

[Tribble 87] E. D. Tribble, M. S. Miller, D.G. Bobrow, C. Abbott, E. Shapiro, Channels:A Generalization of Streams, The Proc. of 4th ICLP. (1987)