

# High-level Specification of Termination Detection in a Distributed System in LOTOS

Tim Mansfield  
Kerry Raymond

Key Centre for Software Technology  
Department of Computer Science  
University of Queensland  
St. Lucia QLD 4067  
AUSTRALIA

Discussion Document No. 1

January 6th, 1989\*

## Abstract

This document is a specification of a system of processes each of which have two states, active and idle. Processes may change state and send messages to each other. The system is said to have terminated when all processes are idle.

The document's purpose is to demonstrate the use of LOTOS for abstract specification of a small distributed system problem.

## 1 LOTOS Specification of the System

This section is a specification, in LOTOS, of a system of communicating processes. Processes have two states, active and idle. The comm gate is the route by which all interprocess communication happens. The term gate is the means by which system detects termination: when all processes are idle, they synchronise on a done message at the term gate.

The specification is of an arbitrary number of processes communicating along the edges of an arbitrary graph, hence the number of processes (`numproc`) and the structure of the graph (`edges`) are passed as parameters to the specification.

```
specification Termination [term, comm] (numproc:nat,  
                                         edges:connections ) :=
```

---

\*printed September 4, 1989

behaviour

```
GenProc [term, comm] (numproc)
||
CommsMedium [term, comm] (edges)
```

where

(\* Some judicious augmentation of the standard types \*)

(\* A 2-Tuple \*)

type Pair is

```
formalsort el1, el2
sorts pair
opns
    <_,_>: el1, el2 -> pair
    1st: pair -> el1
    2nd: pair -> el2
eqns
    forall x:el1, y:el2, z:pair
        ofsort el1
            1st(<x,y>)=x;
        ofsort el2
            2nd(<x,y>)=y;
        ofsort pair
            < 1st(z), 2nd(z) > =z;
```

endtype

type Relation is

```
Sets actualizedby Pair using
sortnames pair for element
renamedby
sortnames relation for set
```

opns

```
dom: relation -> set (* of el1 *)
rng: relation -> set (* of el2 *)
```

eqns

```
forall x:el1, y:el2, z:pair, s1:set, s2:set, r:relation
```

ofsort set

```
dom( {} ) (* domain of empty relation *) = {} (* of el1 *)
dom( Insert( <x,y>, r) ) = Insert(x, dom(r))
```

```
rng( {} ) (* range of empty relation *) = {} (* of el2 *)
rng( Insert( pack(x,y), r) ) = Insert(y, rng(r))
```

endtype

(\* Identifiers for Processes \*)

type Id is Nat\_numbers

renamedby

sortnames ID for nat

eqns

forall x:ID  
ofsort boolean  
x<N = true;

endtype

(\* A Graph of IDs \*)

type Connections is Relation, Boolean

actualizedby Id

sortnames ID for e11

ID for e12

renamedby

sortnames connections for relation

opns

neighbours: connections, ID, ID -> boolean  
members: connections -> IDset  
\_SubSetOf\_: connections, connections -> boolean

eqns

forall x,y:ID, c:connections  
ofsort boolean  
neighbours( c, x, y) = <x,y> IsIn c or <y,x> IsIn c;  
  
{ } SubSetOf c2 = true;  
Insert( <x,y>, c1) SubSetOf c2 =  
neighbours( c2, x, y) and c1 SubSetOf c2;  
  
ofsort IDset  
members( { }) = { };  
members( Insert( <x,y>, r)) =  
Insert( x, Insert(y, members( r)));

endtype

type IdSet is Set

actualizedby Id

sortnames ID for element

renamedby

sortnames IDset for set

endtype

```

(* An enumerated type {active,idle} *)
type Activity is
  sorts activity
  opns active: ->activity
        idle: ->activity
endtype

type MsgType is
  sorts msgtype
  opns
        message: ->msgtype
(* only one message type *)
endtype

(* Permits communication only between processes connected by edges in
the graph *)
process CommsMedium [term, comm] (edges:connections) :=
        comm ?from:ID ?to:ID ?m:msgtype [neighbours(edges, from, to)];
        CommsMedium [comms, term] (edges)
[]
(* terminates along with everything else *)
        term !done; exit

endproc (* CommsMedium *)

(* Generates numproc processes *)
process GenProc [term, comm] (numproc : nat) :=
        [numproc > 1] ->
                InitProc [term, comm] (numproc)
                ||
                GenProc [term, comm] (numproc - 1)
[]
        [numproc = 1] ->
                InitProc [term, comm] (numproc)

endproc (* GenProc *)

(* Initialises processes to be active *)
process InitProc [term, comm] (me: nat) :=
        Proc [term, comm] (me, active)

endproc (* InitProc *)

```

```

(* Models processes, me is the process ID, status is whether the process
is active or idle *)
process Proc [term, comm] (me : nat, status : activity) :=

(* active processes may become idle *)
  [status = active] ->
    i; Proc [term, comm] ( me, idle)
  []

(* active processes may send a message *)
  [status = active] ->
    comm !me ?id:ID !message;
    Proc [term, comm] ( me, active)
  []

(* any process may receive a message, which may make it
active *)
    comm ?id:ID !me !message;
    (
      i; Proc [term, comm] ( me, status)
    []
      i; Proc [term, comm] ( me, active)
    )
  []

(* an idle process is always prepared to terminate if everyone
else agrees *)
  [status = idle] ->
    term !done; exit
  []

(* every process is willing to allow pairs of processes of which
it isn't a member, communicate *)

    comm ?id1:ID ?id2:ID ?m:msgtype [id1 != me and id2 != me];
    Proc [term, comm] (me, status)

endproc (* Proc *)

endspec (* Termination *)

```

## 2 Problems with the Specification

The major problem with the specification is the way in which the graph is modelled. All processes communicate on the comms gate, but the CommsMedium process constrains the passing of messages so that only processes which are connected in the graph can communicate. This has the consequence that every process must be willing to allow communication by other processes to occur (necessitating the final choice in the body of Proc).

This is an awkward way in which to organise communication. Why should processes be aware of communication happening between other processes to whom they are not even connected?

This may not be the way an implementer would prefer to arrange a system of communicating processes which implement this specification. It is likely, that in an implementation, processes would only communicate along links which agree with the edges of the graph. In this case, the last choice in the body of Proc becomes a detail which is necessary for the specification, but not for the implementation. So, the specification actually adds detail, which means that in at least one respect, it is less abstract than it might be.

There appear to be a number of alternative ways to specify a graph of processes in LOTOS. The most obvious is to have a different gate for each communication channel. This is adequate for any *given* number of processes in any *given* graph. However, in an arbitrary graph, processes may differ in the number of channels on which they communicate, making it impossible to model a process in general (since the list of gates is specified when the process is defined, the list is static and bounded).

The solution given above appears to be the only one possible. The group feels that the solution is not abstract and adds unnecessary and irrelevant detail to the specification.