

A Specification in Z of Topor's Termination Detection Algorithm

Kerry Raymond

Tim Mansfield

Key Centre for Software Technology

Department of Computer Science

University of Queensland

St. Lucia QLD 4067

AUSTRALIA

Discussion Document No. 4

January 8th, 1989*

Abstract

Rod Topor's algorithm (in [1]) detects termination in a distributed system of processes which communicate only via message passing without any implicit assumptions of global data.

This specification of Topor's algorithm demonstrates the use of Z for algorithmic specification and represents a refinement of the more abstract specification of Discussion Document No. 3.

Let N be the number of processors, numbered 1 to N .

$$Id \cong 1..N$$

Tokens come in two colours.

$$colour \cong \{white, black\}$$

A Process is considered to be either active or idle.

$$activity \cong \{active, idle\}$$

*printed February 19, 1990

Process

status : *activity*
{ this process's colour }
pcolour : *colour*
{ the set of tokens }
token : *Id* \rightarrow *colour*
hasrepeat : *Boolean*

There are N Processes, arranged in a graph.

Graph

node : *Id* \rightarrow *Process*
edges : *Id* \rightarrow *Id*

{ there are no trivial cycles in the graph }
 $\forall i : Id \bullet (i, i) \notin edges$

{ the graph is connected }
 $\forall i : Id \bullet edges^*(i) = Id$

{ edges are listed one way only }
 $edges \cap edges^{-1} = \{ \}$

On top of the graph, we superimpose a spanning tree.

Tree

Graph
root : *Id*
parent : *Id* \leftrightarrow *Id*
children : *Id* \rightarrow $\mathbb{P} Id$
leaf : $\mathbb{P} Id$

{ Everyone has a parent except root }
 $dom\ parent = Id - \{root\}$

{ the edges of the tree are also part of the graph }
 $parent \subseteq edges \cup edges^{-1}$

{ the edges of the tree all point upward towards the root }
 $\forall i : dom\ parent \bullet$
 $\exists k : 1..N - 1 \bullet$
 $parent^k(i) = root$

{ children are the inverse of the parent }
 $\forall i : Id \bullet$
 $children(i) = \{x : dom\ parent \mid parent(x) = i\}$

{ a leaf doesn't have any children }
 $leaf = \{i : Id \mid children(i) = \{ \}\}$

System

$\Delta Tree$

{ the structures of the graph and tree do not change }

$edges' = edges$

$root' = root$

$parent' = parent$

{ These are derivable }

{ $children' = children$ }

{ $leaf' = leaf$ }

We initialize the System to make all nodes active.

*System*_{INIT}

Tree

$e? : Id \leftrightarrow Id$

$edges = e?$

{ nodes begin active and white without repeats }

{ the leaves have a white token }

$\forall i : Id \bullet$

$i \in leaf \Rightarrow$

$node(i) = (active, white, \{i \mapsto white\}, false)$

$i \notin leaf \Rightarrow$

$node(i) = (active, white, empty, false)$

Active processes send messages

SendMsg

System

$to?, from? : Id$

$node(from?).status = active$

$(from?, to?) \in edges \cup edges^{-1}$

{ A node may become active upon receiving a message }

{ Rule 1. A node which sends a message becomes black }

$(node' = node \oplus \{$

$from? \mapsto (node(from?).status, black, node(from?).token, node(from?).hasrepeat) \} \vee$

$node' = node \oplus \{$

$from? \mapsto (node(from?).status, black, node(from?).token, node(from?).hasrepeat),$

$to? \mapsto (active, node(to?).pcolour, node(to?).token, node(to?).hasrepeat) \})$

An active process can become idle at any time.

Passivate

<i>System</i> $i? : Id$
$node(i?).status = active$ $node' = node \oplus \{i? \mapsto$ $(idle, node(i?).pcolour, node(i?).token, node(i?).hasrepeat)\}$

An idle node with a full complement of tokens, transmits a token to its parent and becomes white and tokenless.

SendToken

<i>System</i> $to?, from? : Id$ $tcolour? : colour$
{ root cannot send a token to its parent } $from? \neq root$
{ the node must be idle } $nodes(from?).status = idle$
{ a leaf node must have its own token } $from? \in leaf \Rightarrow$ $dom(node(from?).token) = \{from?\}$
{ an internal node must have a token from each of its children } $from? \notin leaf \Rightarrow$ $dom(node(from?).token) = children(from?)$
{ the token must be sent to the parent } $to? = parent(from?)$
{ the token sent is black if the node or one of its tokens is black } $tcolour? = black \Leftrightarrow$ $node(from?).pcolour = black \vee$ $black \in rng(node(from?).token)$
{ updating the state of the nodes } $node' = node \oplus \{$ $from? \mapsto (idle, white, node(from?).hasrepeat),$ $to? \mapsto ($ $node(to?).status, node(to?).pcolour,$ $node(to?).token \cup \{from? \mapsto tcolour?\},$ $node(to?).hasrepeat)\}$

On obtaining its full complement of tokens, an active or dirty (directly or via tokens) root initiates a cleansing repeat wave.

GenRepeat

System
$\{ \text{the root has received a token from all of its children } \}$ $dom(node(root).token) = children(root)$
$\{ \text{the root is active or dirty } \}$ $(node(root).status = active \vee$ $node(root).pcolour = black \vee$ $black \in rng(node(root).token))$
$\{ \text{initiate the repeat wave } \}$ $nodes' = node \oplus \{root \mapsto$ $(node(root).status, node(root).pcolour, node(root).token, true)\}$

HandleRepeat

System
$\{ \text{a node must have a repeat in order to handle it } \}$ $\exists i : Id \bullet node(i).hasrepeat$
$\{ \text{a leaf node turns a repeat into a white token } \}$ $i \in leaf \Rightarrow$ $node' = node \oplus \{i \mapsto$ $(node(i).status, white, \{i \mapsto white\}, false)\}$
$\{ \text{a non-leaf node cleans itself and propogates the repeat to its children } \}$ $i \notin leaf \Rightarrow$ $node' = (children(i) \triangleleft node)$ $\oplus \{i \mapsto (node(i).status, white, , false)\}$ $\oplus \{c : children(i) \bullet c \mapsto$ $(node(c).status, node(c).pcolour, node(c).token, true)\}$

The system terminates when the root is idle, white and has a white token from each of its children.

Termination

System
$node(root).status = idle$
$node(root).pcolour = white$
$dom(node(root).token) = children(root)$
$black \notin rng(node(root).token)$

References

- [1] Rodney W. Topor. Termination detection for distributed computations. In *Information*

Processing Letters, pages 33-36, January 1984.