3rd Apr. ‘12

**CMPT 475 Project Distributed Termination Detection**

**Aparna Agarwal (aaa94@sfu.ca, 301087691)**

**Cley Tang (cleyt@sfu.ca, 301116141)**

**James Wall (jwall@sfu.ca, 301089955)**

**Table of Contents**

|  |  |
| --- | --- |
| **Introduction: The Distributed Termination Detection Protocol** | **pg. 3** |
| **Vocabulary used in our model** | **pg. 3** |
| **ASM Diagrams** | **pg. 4** |
| **The Abstract State Machine Model**   * **Abstractions** * **Assumptions** * **Safety and Liveness** | **pg. 9**  **pg. 10**  **pg. 10**  **pg. 10** |
| **Proof that the protocol works** | **pg. 11** |
| **CoreASM Specification** | **pg. 15** |
| **Testing the CoreASM model** | **pg. 19** |
| **Example test runs** | **pg. 20** |

**The Distributed Termination Detection Protocol**

In this project, we have designed an ASM model for Dijkstra’s Termination Detection Algorithm for Distributed Computations, and extended that model to run correctly for multiple computations running on the same distributed network.

Our termination detection protocol follows the same six rules as Dijktra’s algorithm:

1. When active, machine(i+1) keeps the token; when passive, it hands over the token to machine(i)
2. A machine sending a message makes itself black
   1. A machine receiving a message becomes active
3. When machine(i+1) propagates the probe, it hands over a black token to machine(i) if it is black itself, whereas while being white it hands over the color of the token unchanged
4. After the completion of an unsuccessful probe, machine(0) initiates a next probe.
5. Machine(0) initiates a probe by making itself white and sending a white token to machine(i-1)
6. Upon transmission of the token to machine(i), machine(i+1) becomes white. (Note that its original color may have influence the color of the token.)

**Vocabulary used in our model**

Color = {black, white}

Token = {blackToken, whiteToken, noToken}

Machine

Computation

color : Machine X Computation → Color

token : Machine X Computation → Token

terminated : Computation → Boolean

static next : Machine → Machine

monitored isActive : Machine X Computation → Boolean

monitored blackTokenEvent, whiteTokenEvent, sendMessageEvent

**ASM diagrams**

In the proof presented in class, a single computation terminates successfully under the given circumstances. Running multiple computations is no different than running a single computation since computations are treated as if they are independent of each other.

Different aspects of the system, defined by the rules and vocabulary above, are shown in the following state machine diagrams. Represented are diagrams that describe how a machine switches between passive and active, how a regular machine handles tokens, how both a regular machine and a supervisor machine interact with the environment in the context of a single computation, and how the system operates, and ultimately terminates, with multiple computations.



A machine is initially active and spontaneously becomes passive when it finishes a given computation. If a machine is passive, it will change to active if and when a message is received from another machine.



A regular machine receives a token, white or black, from the previous machine in the system. If a machine receives a black token, it will always forward a black token, whether the machine color is white or black. Note: when the machine forwards a black token, the color of the machine changes from black to white. When a machine has a white token, it forwards a white token if its color is white, otherwise if the machine’s color is black, it will forward a black token.



With three binary attributes (white/black, active/passive, token/noToken), there are eight possible states a regular machine can take for a single computation. These states are shown above with all possible transitions between states. The cause of a given transition is listed on the arrow and is determined by the rules set out in the specification.



The key differences between the supervisor machine and the regular machine are in the token handling. Since the supervisor is responsible for starting probes, it always forwards a white token and is always looking for the termination condition based on the token color and its own color when it receives a token. Termination for the individual computation is reached when a white, passive, supervisor machine receives a white token.



The system handles the multiple computations by treating them as separate individual entities. Each computation has its own token, token color, and machine color for each machine. So, while machine 1 might be black for computation 1 it can simultaneously be white for computation 2. In this example, machine 1 would be labeled both black and white, but for different computations. Overall, the supervisor continually initiates probes and terminates each computation separately and concurrently when appropriate, until all computations have reached termination; at this point, the global system termination condition has been reached.

**The Abstract State Machine Model**

ReactOnEvents( m : Machine , c : Computation ) ≡

if blackTokenEvent(m,c) then

token(m,c) := blackToken

if whiteTokenEvent(m,c) then

token(m,c) := whiteToken

if sendMachineEvent(m,c) then

color(m,c) := black

InitializeMachine ( m : Machine, c : Computation ) ≡

token(m,c) := noToken

color(m,c) := white

RegularMachineProgram ( m : Machine ) ≡

(∀ c ∈ Computation with ¬terminated(c) )

ReactOnEvents(m,c)

if ¬isActive(m,c) ∧ ¬token(m,c)=noToken

InitializeMachine(m,c)

if color(m,c) = black

ForwardToken(blackToken, nextMachine(m), c)

else if color(m,c) = white

ForwardToken(token(m,c), nextMachine(m), c)

SupervisorMachineProgram ( m : Machine ) ≡

(∀ c ∈ Computation with ¬terminated(c) )

ReactOnEvents(m,c)

if ¬isActive(m,c) ∧ ¬token(m,c)=noToken

terminated(c) := true

if (terminated(c) for ∀ c ∈ Computation)

ReportGlobalTermination

else

InitializeMachine(m,c)

ForwardToken(m, whiteToken, nextMachine(m), c)

Initial State ≡

(∀ c ∈ Computation)

terminated(c) := false

(∃ machine0 ∈ Machine) (program(machine0) = SupervisorMachineProgram) ∧ token(machine0) = blackToken) ∧

(∀ m ∈ Machine) (m≠machine0 ⇒ program(m) = RegularMachineProgram)

(∀ m ∈ Machine) (color(m) = white)

* The above ASM model abstractly models the generalized version of our protocol and its requirements. Each computational agent (regular or supervisor machine) is assigned a machine program in the above model.

**Abstractions**

The following have been left abstract in our ASM model:

* Operations
  + ForwardToken (t : Token, m : Machine, c : Computation)
  + ReportGlobalTermination
* Tables (static functions)
  + static next : Machine → Machine
* Interfaces (monitored functions)
  + blackTokenEvent : Machine X Computation → Boolean
  + whiteTokenEvent : Machine X Computation → Boolean
  + sendMessageEvent : Machine X Computation → Boolean
  + isActive : Machine X Computation → Boolean
* Computation processing

**Assumptions**

We make the following assumptions about our system model with regards to its interactions with the environment, and the initial machine state.

* **Interactions with the environment**
  + Passing tokens, sending messages, becoming active/passive happen immediately (each event incurs no lag time)
  + Computations take random amounts of time to complete
  + Machines send messages with random probability
  + No concept of data concurrency control
  + Each token is held by only one machine at a time at any given instant
  + Computations are independent of each other
* **Initial machine state**
  + All machines are initially active
  + All computations being considered are initially running
  + Each machine is initially white
  + The Supervisor machine initially has a black token, and regular machines have no token

**Safety and Liveness**

The protocol ensures that:

- The system is terminated only when all machines are passive, and no machine is running any computation (safety)

- The machines forward tokens as soon as they become passive, without any delay (liveness)

**Proof that the protocol works**

In order to grasp why this system works for all cases we must again refer back to the 6 basic rules of the system.

1. *Rule:* When active, *machine i*+1 keeps the token; when passive, it hands over the token to *machine i*.
2. *Rule*: A machine sending a message makes itself black.
3. *Rule*: When *machine i*+1 propagates the probe, it hands over a black token to *machine i* if it is black itself, whereas while being white it leaves the colour of the token unchanged.
4. *Rule*: After the completion of an unsuccessful probe, *machine* 0 initiates a next probe.
5. *Rule*: *Machine* 0 initiates a probe by making itself white and sending a white token to *machine i*-1.
6. *Rule*: Upon transmission of the token to *machine i*, *machine i*+1 becomes white. (Note that its original colour may have influenced the colour of the token.)

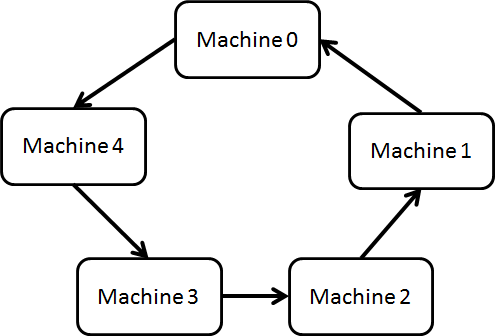
There are 3 cases that this system must handle:

1. No machines are sending or receiving messages
2. Machines are sending messages (or become active) during the probing
3. Multiple computations on each machine along with Case 1 and 2

Keep in mind this algorithm runs in real-time. The algorithm probes the system by propagating a white token through the system. If the token is propagated through the system and remains white, we know that all machines are inactive and we can terminate the entire distributed system.

**Case 1**

For this example we will consider a simple system with 5 machines as seen below.



From the diagram it is quite easy to see that passing a white token through the system will reach the end white (Rule 3). Provided that no machines send any messages, the system will terminate. We will examine this in more detail.

We start with Rule 5. Machine 0 initiates the probe by making itself white. Since none of the machines are sending messages, all machines remain white. The white token is passed from Machine 0 to Machine 4.

Since Machine 4 is white, it passes on the white token to Machine 3 as stated in Rule 3. Similarly this will happen for each subsequent machine until the token reaches Machine 0. Since each machine cannot change the colour of the token, the token remains white. Machine 0 receives the white token and then the whole system can be terminated.

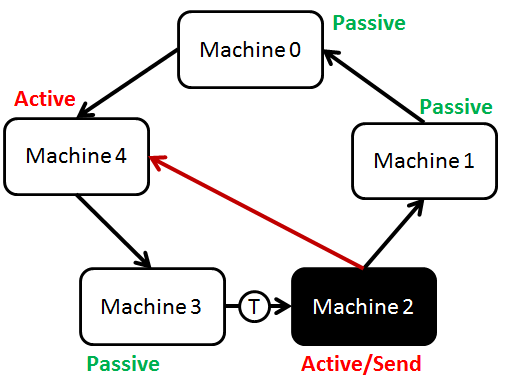
In summary, the white token passes through the white machines without changing colour. When it reaches the end, the system terminates.

**Case 2**

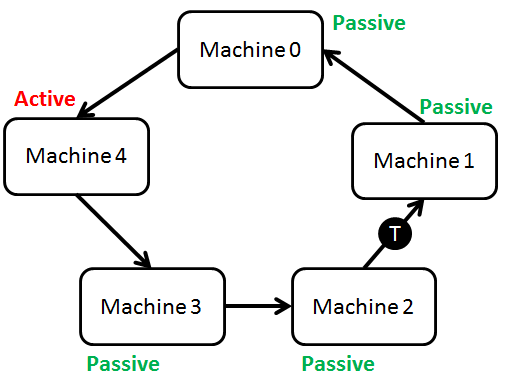
How about the case where messages are sent during the probing?

We consider again the same diagram as shown above, suppose that Machine 4 sends a message. Rule 2 would make this machine then turn black. When the token is passed from Machine 0 to Machine 4 (Rule 2), Machine 4 holds the token until it becomes passive and turns white (Rule 1). Rule 3 would make the token Black. At this point the token will not become white until it reaches Machine 0. Hence, the system will not terminate at the end of this first probe.

Now Rule 4 comes in to effect, the system is probed again. Let’s also suppose the white token reaches Machine 3 and Machine 2 decides to send a message before the token reaches it.

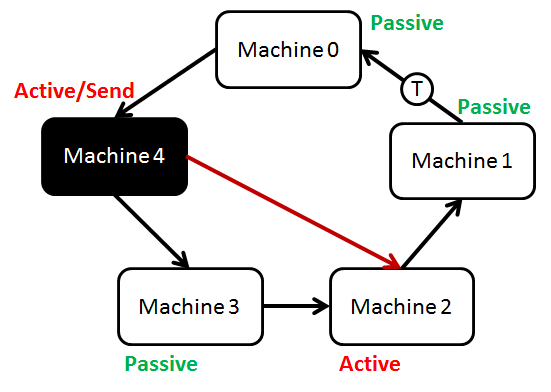


Again, the token will turn black after Machine 2 passes it along, turns white (Rule 6) and the whole process is repeated again.



Notice how Machine 4 is active but did not turn black. The token will eventually reach   
Machine 4. As long as Machine 4 does not send a message, the token will remain white and reach Machine 0 again thus terminating the whole system.

But what if we have this situation?



Machine 0 will theoretically receive a white token and terminate with Machine 4 still active! However, this situation is impossible because of Rule 1 and Rule 6. The system would have to become passive and white in order to pass the token along.

Furthermore, only active machines can send messages. So only machines in the token’s path will be able to send messages and turn black.

**Case 3**

So far we have only seen the system operate with each machine processing 1 computation at a time. Now, consider the case where each machine can have multiple computations along with the above 2 cases.

Suppose our machines can have n number of simultaneous computations, our algorithm will be slightly modified so that we have n tokens that probe the system independently from each other. All of the n tokens have to be white before they reach Machine 0. When all of the tokens return to Machine 0 white, the system can terminate.

Therefore the system works for all cases.

**CoreASM Specification**

// CMPT 475 Course Project (Spring, '12)

// Aparna Agarwal, 301087691

// Cley Tang, 301116141

// James Wall, 301089955

**CoreASM** DistributedTerminationDetection

**use** StandardPlugins

**use** TimePlugin

**use** MathPlugin

**enum** Colour = {black, white}

**enum** Token = {noToken, blackToken, whiteToken}

**enum** Machine = {machine0, machine1, machine2, machine3, machine4, machine5, machine6, machine7, machine8}

**enum** Computation = {comp1, comp2, comp3}

**function** colour : Machine \* Computation -> Colour

**function** token : Machine \* Computation -> Token

**function** nextMachine: Machine -> Machine

**function** isActive : Machine \* Computation -> boolean

**function** blackTokenEvent : Machine \* Computation -> boolean

**function** whiteTokenEvent : Machine \* Computation -> boolean

**function** sendMessageEvent : Machine \* Computation -> boolean

**function** terminated : Computation -> boolean

**universe** Agents = {supervisorMachine, regularMachine}

**init** InitRule

**rule** InitRule = **seqblock**

startTime := now

**forall** c **in** Computation **do**

**seqblock**

terminated(c) := **false**

**forall** m **in** Machine **do**

**seqblock**

**if** m = machine0 **then** **seqblock**

InitializeMachine(m, c)

token(m, c) := blackToken

AssignNextMachine(m)

**endseqblock**

**else** **par**

InitializeMachine(m, c)

AssignNextMachine(m)

**endpar**

isActive(m, c) := **true**

**endseqblock**

PrintProgram(c)

**endseqblock**

program(supervisorMachine) := @SupervisorMachineProgram

program(regularMachine) := @RegularMachineProgram

program(**self**) := **undef**

**endseqblock**

**rule** PrintProgram(c) = **seqblock**

**print** "Time: " + ((now - startTime) / 1000) + " seconds:"

**choose** m **in** Machine **with** token(m, c) != noToken **do** **seqblock**

**print** "Computation " + c + ":"

**print** "Machine " + m + " holding " + token(m, c)

**print** "Machine " + m + ":- colour: " + colour(m, c) + " , Active: " + isActive(m, c)

**print** ""

**endseqblock**

**endseqblock**

**rule** AssignNextMachine(m) = **par**

**if** m = machine1 **then**

nextMachine(m) := machine0

**if** m = machine2 **then**

nextMachine(m) := machine1

**if** m = machine3 **then**

nextMachine(m) := machine2

**if** m = machine4 **then**

nextMachine(m) := machine3

**if** m = machine5 **then**

nextMachine(m) := machine4

**if** m = machine6 **then**

nextMachine(m) := machine5

**if** m = machine7 **then**

nextMachine(m) := machine6

**if** m = machine8 **then**

nextMachine(m) := machine7

// Add more statements here to add new machines to the network

// Assign the last machine as machine0's nextMachine

**if** m = machine0 **then**

nextMachine(m) := machine8

**endpar**

**rule** ForwardToken(t, m, c) = **seqblock**

token(m, c) := t

**if** t = blackToken **or** colour(m, c) = black **then**

blackTokenEvent(m, c) := **true**

**else**

whiteTokenEvent(m, c) := **true**

PrintProgram(c)

**endseqblock**

**rule** ReactOnEvents(m, c) = **par**

**if** blackTokenEvent(m, c) **then**

token(m, c) := blackToken

**if** whiteTokenEvent(m, c) **then**

token(m, c) := whiteToken

**if** sendMessageEvent(m, c) **then** **seqblock**

colour(m, c) := black

**choose** mac1 **in** Machine **with** m != mac1 **do** **seqblock**

**if** (**not** isActive(mac1, c)) **then**

isActive(mac1, c) := **true**

**endseqblock**

**endseqblock**

**endpar**

**rule** InitializeMachine(m, c) = **seqblock**

token(m, c) := noToken

colour(m, c) := white

blackTokenEvent(m, c) := **false**

whiteTokenEvent(m, c) := **false**

sendMessageEvent(m, c) := **false**

**endseqblock**

**rule** RegularMachineProgram =

**forall** c **in** Computation **with** (**not** terminated(c)) **do**

**par**

**forall** m **in** Machine **with** m != machine0 **do**

**par**

**seq**

ReactOnEvents(m, c)

**next**

**if** (**not** isActive(m, c)) **and** token(m, c) != noToken **then** **seqblock**

col(c) := colour(m, c)

tok(c) := token(m, c)

InitializeMachine(m, c)

**if** col(c) = black **then**

ForwardToken(blackToken, nextMachine(m), c)

**else** **if** col(c) = white **then**

ForwardToken(tok(c), nextMachine(m), c)

**endseqblock**

**else**

ActiveCondition(m, c)

**endpar**

**endpar**

**rule** SupervisorMachineProgram = **seqblock**

**if** (**forall** c **in** Computation **holds** terminated(c)) **then** **par**

ReportGlobalTermination

**endpar**

**forall** c **in** Computation **with** (**not** terminated(c)) **do**

**par**

**choose** m **in** Machine **with** m = machine0 **do**

**par**

**seq**

ReactOnEvents(m, c)

**next**

**if** (**not** isActive(m, c)) **and** token(m, c) != noToken **then** **par**

**if** colour(m, c) = white **and** token(m, c) = whiteToken **then** **seqblock**

terminated(c) := **true**

**print** "Computation " + c + " terminated\n"

**endseqblock**

**else** **seqblock**

InitializeMachine(m, c)

ForwardToken(whiteToken, nextMachine(m), c)

**endseqblock**

**endpar**

**else**

ActiveCondition(m, c)

**endpar**

**endpar**

**endseqblock**

**rule** ActiveCondition(m, c) = **seqblock**

**if** isActive(m, c) **then** **seqblock**

rand := random

**if** rand < 0.1 **then** **par**

sendMessageEvent(m, c) := **true**

**endpar**

**if** rand > 0.75 **then** **seqblock**

isActive(m, c) := **false**

**if** token(m, c) != noToken **then**

PrintProgram(c)

**endseqblock**

**endseqblock**

**endseqblock**

**rule** ReportGlobalTermination = **seqblock**

program(supervisorMachine) := **undef**

program(regularMachine) := **undef**

program(**self**) := **undef**

**print** "Global Termination"

**endseqblock**

**Testing the CoreASM Model**

We tested our CoreASM specification with a varied number of machines and computations.

In the submission made to the TA are included the complete outputs for the following test runs:

* 8 machines, running 3 computations
* 10 machines, running 3 computations
* 10 machines, running 4 computations
* 9 machines, running 5 computations
* 8 machines, running 3 computations (boundary case, where the system terminates in one probe)

In order to repeat these experiments, the following modifications need to be made to the CoreASM specification:

* Add/remove machines from the enum Machine (line#15):

**enum** Machine = {machine0, machine1, machine2, machine3, machine4, machine5, machine6, machine7, machine8}

* Add/remove computations from the enum Computation (line#16):

**enum** Computation = {comp1, comp2, comp3}

* Modify the rule AssignNextMachine(m) (line# 73) according to the modified Machine.

For example, say we want to add a new machine (machine9).

* Our enum Machine becomes:

**enum** Machine = {machine0, machine1, machine2, machine3, machine4,

machine5, machine6, machine7, machine8, machine9}

* The new last machine would become Machine0’s next machine, and the new machine would need to be assigned a next machine too.

Our AssignNextMachine(m) rule would then be modified to:

**rule** AssignNextMachine(m) = **par**

**………**

**………**

**if** m = machine8 **then**

nextMachine(m) := machine7

// Add more statements here to add new machines to the network

**if** m = machine9 **then**

nextMachine(m) := machine8

// Assign the last machine as machine0's nextMachine

**if** m = machine0 **then**

nextMachine(m) := machine9

**endpar**

**Test Runs**

1. **8 machines, running 3 computations**

Time: 0 seconds:

Computation comp2:

Machine machine0 holding blackToken

Machine machine0:- colour: white , Active: true

Time: 0 seconds:

Computation comp1:

Machine machine0 holding blackToken

Machine machine0:- colour: white , Active: true

Time: 0 seconds:

Computation comp3:

Machine machine0 holding blackToken

Machine machine0:- colour: white , Active: true

Time: 0.241 seconds:

Computation comp2:

Machine machine0 holding blackToken

Machine machine0:- colour: white , Active: false

Time: 0.241 seconds:

Computation comp1:

Machine machine0 holding blackToken

Machine machine0:- colour: white , Active: false

Time: 0.241 seconds:

Computation comp3:

Machine machine0 holding blackToken

Machine machine0:- colour: white , Active: false

Time: 0.664 seconds:

Computation comp3:

Machine machine8 holding whiteToken

Machine machine8:- colour: white , Active: true

……..

………

………

Time: 28.62 seconds:

Computation comp3:

Machine machine0 holding whiteToken

Machine machine0:- colour: white , Active: false

Computation comp3 terminated

………

……….

Time: 29.486 seconds:

Computation comp2:

Machine machine0 holding whiteToken

Machine machine0:- colour: white , Active: false

Computation comp2 terminated

……..

……..

Time: 32.197 seconds:

Computation comp1:

Machine machine1 holding whiteToken

Machine machine1:- colour: white , Active: false

Time: 32.402 seconds:

Computation comp1:

Machine machine0 holding whiteToken

Machine machine0:- colour: white , Active: false

Computation comp1 terminated

Global Termination

1. **10 machines, running 4 computations**

Time: 0 seconds:

Computation comp2:

Machine machine0 holding blackToken

Machine machine0:- colour: white , Active: true

Time: 0 seconds:

Computation comp1:

Machine machine0 holding blackToken

Machine machine0:- colour: white , Active: true

Time: 0 seconds:

Computation comp3:

Machine machine0 holding blackToken

Machine machine0:- colour: white , Active: true

Time: 0 seconds:

Computation comp4:

Machine machine0 holding blackToken

Machine machine0:- colour: white , Active: true

……….

……….

Time: 3.675 seconds:

Computation comp4:

Machine machine0 holding blackToken

Machine machine0:- colour: black , Active: false

Time: 3.675 seconds:

Computation comp1:

Machine machine0 holding blackToken

Machine machine0:- colour: black , Active: false

Time: 3.884 seconds:

Computation comp4:

Machine machine10 holding whiteToken

Machine machine10:- colour: black , Active: false

Time: 3.884 seconds:

Computation comp3:

Machine machine10 holding whiteToken

Machine machine10:- colour: black , Active: false

………

………

Time: 15.248 seconds:

Computation comp3:

Machine machine0 holding whiteToken

Machine machine0:- colour: white , Active: false

Computation comp3 terminated

……….

……….

Time: 15.888 seconds:

Computation comp1:

Machine machine0 holding whiteToken

Machine machine0:- colour: white , Active: false

Computation comp1 terminated

……….

……….

Time: 22.691 seconds:

Computation comp2:

Machine machine0 holding whiteToken

Machine machine0:- colour: white , Active: false

Computation comp2 terminated

……….

……….

Time: 27.626 seconds:

Computation comp4:

Machine machine0 holding whiteToken

Machine machine0:- colour: white , Active: false

Computation comp4 terminated

Global Termination