Egon Börger (Pisa) & Alexander Raschke (Ulm)

Termination Detection for Distributed Runs

Illustrating the role of model validation

Università di Pisa, Dipartimento di Informatica boerger@di.unipi.it
Universität Ulm, Abteilung Informatik alexander.raschke@uni-ulm.de

See Ch. 3.2 of Modeling Companion

Termination detection for distributed computations

Goal: illustrate the *role of mechanical model validation* (besides inspection)

- for discovery of implicit assumptions and potential conflicts in requirements
  - in 'the virtuous circle where difficulties with the formal specification prompt further elicitation of the requirements' (Gervasi/Riccobene, Dagstuhl 2014)

Exl: Algorithm proposed by Dijkstra et al. (Inf.Proc.Lett. = EWD 840)

- 'to demonstrate how the algorithm can be derived in a number of steps'

We rephrase the requirements from EWD 840 and follow the analysis performed by Gervasi/Riccobene (see references below).

Plant&FunctionalRequirement

Plant&FunctionalReq. We consider N machines arranged in a ring. Each machine is either active or passive. The state in which all machines are passive is stable: the distributed computation is said to have terminated. The purpose of the algorithm to be designed is to enable one of the machines, machine nr. 0 say, to detect that this stable state has been reached.

Signature reflecting these elements:

- static finite set $Machine = \{m_0, \ldots, m_{N-1}\}$ with designated $master = m_0$. We write $number(m_i) = i \bmod N$.
- static ring (background) structure: $pred : Machine \to Machine$ with $pred(m_{i+1}) = m_{i \bmod N}$
- $mode \in \{active, passive\}$
- *Stable* **iff forall** $m \in Machine\ mode(m) = passive$

ProbeSignatureRequirement

ProbeSignatureReq. Denote the process by which termination is to be detected by the "probe". We assume the availability of communication facilities such that (i) machine nr. 0 can initiate the probe by sending a signal to machine nr. $N-1$ (ii) machine nr. $i+1$ can propagate the probe around the ring by sending a signal to machine nr. i. The propagation of the probe around the ring allows us to describe that probe as sending a token around the ring. The probe ends with machine nr. 0 being the machine at which the token resides. The token being returned to machine nr.0 will be an essential component of the justification of the conclusion that all machines are passive.

Signature: *hasToken* ('the machine at which the token resides')=true

FORWARDTOKEN = -- 'sending a signal' 'hands over the token'

$hasToken(\textbf{self}) := false$ -- from machine nr. $i+1$

$hasToken(pred(\textbf{self})) := true$ -- to machine nr. i

TokenProgressRequirement

TokenProgressReq. When active, machine nr. $i+1$ keeps the token; when passive, it hands over the token to machine nr. i. For each machine, the transition from the active to the passive state may occur "spontaneously".

NB. $i+1$ presumably refers to any machine except the $master$.
Tentative description (see below the final definition):

PassToken' = -- NB. to be extended below

  if $mode = passive$ **and** **self** $\neq master$ **and** $hasToken$

    then ForwardToken

BecomePassive' = SpontaneouslyDo

  (**if** $mode = active$ **then** $mode := passive$)

    where

    $\mathrm{SpontaneouslyDo}(\mathcal{M}) =$ **choose** $\mathcal{R} \in \mathcal{M} \cup \{$**skip**$\}$ **do** $\mathcal{R}$

ActivationRequirement: some questions (1)

ActivationReq. Only active machines send so-called "messages" to other machines; msg transmission is considered instantaneous. After having received a msg, a machine is active; the receipt of a msg is the only mechanism that triggers for a passive machine its transition to activity.

Qu. 1: What to do with concurrently arriving multiple msgs?

- to be treated as multiple triggers, to be elaborated one after the other?
- to be collapsed into one cumulative trigger?

We guess (and assume): one trigger. Then the effect to receive a sent msg via an 'instantaneous transmission' could be modeled abstractly as follows:

$$\textsc{SendMsg'}(dest) = (hasMessage_{dest} := true)$$

But see below for yet another problem with ActivationReq.

ActivationRequirement: some questions (2)

Qu.2: Are msgs consumed and when?

Presumably yes, namely when becoming active.

Qu.3:

- Constraint to 'only active machines'?
- Msg receipt 'the only mechanism that triggers ... transition to activity'?

Presumably these are properties stated for a declarative concern, to describe the program axiomatically.

In the ASM model these two properties become true by definition.

Tentative description of ActivationRequirement

TriggerActivation' =

if $mode = active$ **then** SpontaneouslyDo

choose $dest \in Machine$ SendMsg'$(dest)$

BecomeActive' =

if $hasMessage$ **then**

$mode := active$

$hasMessage := false$ -- consume trigger

Conflict: If at machine m, upon becoming active, a new 'instantaneous' msg arrives, sent by another machine's SendMsg'(m) action?

That requires simultaneous update of $hasMessage_m$ to true and false.

Resolving the ActivationReq conflict

By giving priority to activation. This can be modeled easily by *update instructions*.

Here we use two *update instructions* $\textsc{Activate}(trigger)$ and $\textsc{Deactivate}(trigger)$ with the following behavior:

- simultaneously executed $\textsc{Activate}(trigger)$ actions, even if in presence of a simultaneously to be executed $\textsc{Deactivate}(trigger)$ action, produce the single update $trigger := true$
- $\textsc{Deactivate}(trigger)$ executed alone updates $trigger$ to $false$

Update instructions are supported in CoreASM by plug-ins

Message sending with resolved conflict

TriggerActivation' = -- to be refined below by 'coloring'

  if $mode = active$ **then** SpontaneouslyDo

    choose $dest \in Machine$ Activate($Triggered_{dest}$)

Define derived fct: $hasMessage$ **iff** $Triggered_{\textbf{self}} = true$

BecomeActive =

  if $hasMessage$ **then**

    $mode := active$

    Deactivate($Triggered$) -- consume this trigger

Further requirements needed

Pbl. Machines can be (re-) activated after having handed over the token to their predecessor

- some mechanism needed to inform the token owner about the probe failure resulting from such an activation
  - so that this information is available when the token comes back to the master

This information is provided by black coloring of activators and of tokens they forward, such that:

- a token, once colored black, remains black until the end of the probe, where the $master$ recognizes this as probe failure

Reactivation coloring requirements (1)

RecordActivationEventReq. Machines and tokens are postulated to be either black or white. A machine sending a message to a recipient with a number higher than its own makes itself black.

The first clause requires a signature extension:

$color(m), tokenColor \in \{black, white\}$ for $m \in Machine$

The second clause requires an extension of $\textsc{Activate}(Triggered_m)$ by the following update:

if $number(m) > number(\textbf{self})$ **then** $color(\textbf{self}) := black$

Reactivation coloring requirements (2)

ForwardActivationEventInfoReq. When machine nr. $i+1$ propagates the probe, it hands over a black token to machine nr. i if it is black itself, whereas while being white it leaves the colour of the token unchanged. Upon transmission of the token to machine nr. i, machine nr. $i+1$ becomes white. (Note that its original colour may have influenced the colour of the token).

These two clauses require an extension of the above defined PassToken' by updates concerning the coloring of tokens and machines.

But before proceeding to this refinement, let us analyse a consistency issue concerning the preceding two coloring requirements.

Possible conflict bw msg sending and probe propagation

- RecordActivationEventReq stipulates that in certain cases a message sender machine must make itself black.
- ForwardActivationEventInfoReq stipulates that a probe propagating machine must make itself white.

A conflict shows up if a machine simultaneously sends a msg to a machine with a higher number and propagates the probe.

Conflict resolution possible in many ways:

- sequentialization of the two actions
- assuming interleaving semantics: either a msg is sent or the probe is propagated
- allow a machine to SendMessage only when it does not have the token (solution adopted here)

Resolving msgSending/probePropagation conflict

$\textsc{TriggerActivation} =$ // see *ActivationReq*

  if $mode = active$ **then** $\textsc{SpontaneouslyDo}(\textsc{TriggerAct})$

where

  $\textsc{TriggerAct} =$ -- 'instantaneous msg transmission'

    if not $hasToken$ **then** -- coloring conflict resolving guard

      choose $dest \in Machine$ **do** $\textsc{SendMessage}(dest)$

  $\textsc{SendMessage}(m) =$

    $\textsc{Activate}(Triggered(m))$

    if $number(m) > number(\textbf{self})$ **then** $color(\textbf{self}) := black$

    -- see *RecordActivationEventReq*

NB. In ASM model there is no need to stipulate for a machine that 'while being white it leaves the colour of the token unchanged', as needed in a declarative formulation.

Conflict between TokenProgressReq and ActivationReq

- by TokenProgressReq, 'the transition from the active to the passive state may occur "spontaneously"'
- by ActivationReq, if a machine $hasMessage$ it should BecomeActive

Possible conflict resolutions:

- interleaving semantics for concurrent processes: a machine either elaborates a received msg or spontaneously becomes $passive$
- priority to env stimuli: stipulate that machines can become passive only when they are not triggered by the environment

BecomePassive = **if not** $hasMessage$ **then**

SpontaneouslyDo

if $mode = active$ **then** $mode := passive$

ProbeStartRequirement

ProbeStartReq. Machine nr. 0 initiates the probe by making itself white and sending a white token to machine nr. $N-1$. After the completion of an unsuccessful probe, machine nr. 0 initiates the next probe.

'sending a white token' and the ForwardActivationEventInfoReq, whereby

> machine nr.$i+1$ ... hands over a black token ... if it is black itself

require first of all to add coloring updates to $\textsc{ForwardToken}$:

$\textsc{ColorToken} =$

if **self** $= master$ **then** $tokenColor := white$

else if $color($**self**$) = black$ **then** $tokenColor := black$

Furthermore, the master (as every token passing machine, by ForwardActivationEventInfoReq) must $\textsc{GetWhite}$.

ColorForwardToken

ColorForwardToken =

  ColorToken

  ForwardToken

  GetWhite

where

  ColorToken =

    if **self** = *master* **then** *tokenColor* := *white*

      else if *color*(**self**) = *black* **then** *tokenColor* := *black*

  GetWhite = (*color*(**self**) := *white*)

PassToken' completion by coloring

PassToken' is extended by coloring updates as follows:

PassToken =

if $hasToken$ **and** $mode = passive$ **then**

if **self** $\neq master$ **then** Color&ForwardToken

NB. The ForwardActivationEventInfoReq stipulates:

'Upon transmission of the token to machine nr. i, machine nr. $i+1$ becomes white.'

Probe Start and Termination

EvalActivityInfoReq. When a black token is returned to machine nr. 0 or the token is returned to a black machine nr. 0, the conclusion of termination cannot be drawn.

This requirement is expressed as follows:

$ProbeNotSuccessful$ **iff**

$hasToken(master)$ **and**

$(tokenColor = black$ **or** $color(master) = black)$

The ProbeStartReq that 'After the completion of an unsuccessful probe, machine nr. 0 initiates the next probe' is expressed as follows:

$\textsc{StartProbe} =$

if $ProbeNotSuccessful$ **and** **self** $= master$ **then**

$\textsc{Color\&ForwardToken}$

InitializationRequirement

InitializationReq. It is furthermore required that the detection algorithm can cope with any distribution of the activity at the moment machine nr. 0 initiates the detection algorithm.

NB. There should also be $NoMsgsAround$.

A state is an initial state iff it satisfies the following:

forall $m \in Machine$

$mode(m) \in \{active, passive\}$ -- *InitializationReq*

$color(m) \in \{black, white\}$ -- any initial machine coloring

$hasToken(master) = true$ -- the master $hasToken$

if $m \neq master$ **then** $hasToken(m) = false$ -- ... and nobody else

$Triggered(m) = false$ -- meaning: NoMsgsAround

$tokenColor = black$ -- triggers $master$ to start a probe

The final concurrent ASM

$\textsc{DistrTerminationDetect} =$

$(m, \textsc{TerminationDetection}_m)_{m \in Machine}$

Each machine has an instance of the same program, namely:

$\textsc{TerminationDetection} =$

- $\textsc{StartProbe}$ -- done by $master$ if $ProbeNotSuccessful$
- $\textsc{TriggerActivation}$ -- spontaneously if $active$ without token
- $\textsc{BecomeActive}$ -- if $hasMessage$
- $\textsc{BecomePassive}$ -- spontaneously if **not** $hasMessage$
- $\textsc{PassToken}$ -- if $passive$ and $\neq master$

Functional correctness

$TerminationDetected$ **iff**

$hasToken(master)$ **and** $mode(master) = passive$

and $color(master) = white$ **and** $tokenColor = white$

Proposition: In every concurrent run of DistrTerminationDetect, started in an initial state, if $TerminationDetected$ becomes true in some state, then this state is $stable$ (meaning that all machine are $passive$).

For a proof (in need of an additional machine eagerness assumption which is not mentioned in the paper, but results from the instantaneous message transmission assumption) see ModelingBook Ch.3.2.3

Final observation and References

CoreASM executed version revealed some of the inconsistencies and missing assumptions.

- Vincenzo Gervasi and Elvinia Riccobene: *From English to ASM*: On the process of deriving a formal specification from a natural language one
  - Nov. 8, 2013 (Dagstuhl Seminar 13372)
    DOI: 10.4230/DagRep.3.9.74
    URL: http://drops.dagstuhl.de/opus/volltexte/2014/4358/
- Original article: Edsger W.Dijkstra, W.H.J.Feijen and A.J.M. van Gasteren: *Derivation of a termination detection algorithm for distributed computations*
  - Information Processing Letters 16: 217-219 (1983)=EWD840
- E. Börger and A. Raschke: Modeling Companion for Software Practitioners. Springer 2018
  http://modelingbook.informatik.uni-ulm.de

Copyright Notice