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** $\text{Machine} = \{m_0, \ldots, m_{N-1}\}$ with designated master $= m_0$. We write $\text{number}(m_i) = i \mod N$.
- **static ring (background) structure**: $\text{pred} : \text{Machine} \rightarrow \text{Machine}$ with $\text{pred}(m_{i+1}) = m_i \mod N$
- **mode** $\in \{\text{active, passive}\}$
- **Stable iff** forall $m \in \text{Machine}$ $\text{mode}(m) = \text{passive}$
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: \( \text{hasToken} \) (‘the machine at which the token resides’)\( = \)true

\[
\text{FORWARD_TOKEN} = \quad \text{-- ‘sending a signal’ ‘hands over the token’}
\]
\[
\text{hasToken}(\text{self}) := \text{false} \quad \text{-- from machine nr. } i + 1
\]
\[
\text{hasToken}(\text{pred}(\text{self})) := \text{true} \quad \text{-- to machine nr. } i
\]
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):

\[
\text{PassToken'} = \begin{cases} 
\text{ForwardToken} & \text{if mode = passive and self \neq master and hasToken} \\
\text{SpontaneouslyDo} & \text{BecomesPassive'} = \begin{cases} 
\text{SpontaneouslyDo} & \text{if mode = active then mode := passive} \\
\text{where} \\
\text{SpontaneouslyDo}(\mathcal{M}) = \text{choose } \mathcal{R} \in \mathcal{M} \cup \{\text{skip}\} \text{ do } \mathcal{R} 
\end{cases}
\end{cases}
\]
**Activation Requirement: 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:

\[ \text{SendMsg}'(dest) = (hasMessage_{dest} := \text{true}) \]

But see below for yet another problem with ActivationReq.
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’** =
\[
\text{if } mode = \text{ active then } \text{SpontaneouslyDo}
\]
\[
\text{choose } dest \in \text{Machine SENDMsg’}(dest)
\]

**BecomeActive’** =
\[
\text{if hasMessage then}
\]
\[
mode := \text{ active}
\]
\[
hasMessage := \text{ false}
\]
\[
\text{-- consume trigger}
\]

**Conflict:** If at machine \( m \), upon becoming active, a new ‘instantaneous’ msg arrives, sent by another machine’s \( \text{SENDMsg’}(m) \) 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 Activate(\textit{trigger}) and Deactivate(\textit{trigger}) with the following behavior:

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

Update instructions are supported in CoreASM by plug-ins
TriggerActivation =  

\[
\text{if } \text{mode} = \text{active} \text{ then } \text{SpontaneouslyDo}
\]

\[
\text{choose dest } \in \text{Machine Activate}(\text{Triggered}_{\text{dest}})
\]

Define derived fct: \( \text{hasMessage iff } \text{Triggered}_{\text{self}} = \text{true} \)

BecomeActive = 

\[
\text{if hasMessage then}
\]

\[
\text{mode} := \text{active}
\]

Deactivate(\text{Triggered})

\[\text{-- consume this trigger}\]
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
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:

\[ \text{color}(m), \text{tokenColor} \in \{\text{black}, \text{white}\} \text{ for } m \in \text{Machine} \]

The second clause requires an extension of \text{Activate(Triggered}_m) by the following update:

\[ \text{if } \text{number}(m) > \text{number}(\text{self}) \text{ then } \text{color}(\text{self}) := \text{black} \]
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 \texttt{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

\textbf{TriggerActivation} = \begin{verbatim}
  // see ActivationReq
  if mode = active then SpontaneouslyDo(TriggerAct)
\end{verbatim}

where

\textbf{TriggerAct} =
\begin{verbatim}
  if not hasToken then -- coloring conflict resolving guard
    choose dest \in Machine do SendMessage(dest)
\end{verbatim}

\textbf{SendMessage}(m) =
\begin{verbatim}
  Activate(Triggered(m))
  if number(m) > number(self) then color(self) := black
\end{verbatim}

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** = \( \text{if not } hasMessage \text{ then} \)

**SpontaneouslyDo**
\[
\text{if mode }= \text{ active then mode } := \text{ passive}
\]
**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 **ForwardToken**: 

```
COLOR_TOKEN =
    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 **GET WHITE**.
**Color&ForwardToken**

\[
\text{Color&ForwardToken} =
\]

\[
\text{ColorToken} \quad \text{ForwardToken} \
\]

\[
\text{GetWhite}
\]

**where**

\[
\text{ColorToken} =
\]

\[
\text{if self} = \text{master} \text{ then } \text{tokenColor} := \text{white}
\]

\[
\text{else if color(self) = black then tokenColor := black}
\]

\[
\text{GetWhite} = (\text{color(self) := white})
\]
PassToken’ completion by coloring

PassToken’ is extended by coloring updates as follows:

\[
\text{PassToken} =
\begin{align*}
\text{if } & \text{hasToken and mode = passive then} \\
\text{if } & \text{self} \neq \text{master then Color\&ForwardToken}
\end{align*}
\]

NB. The ForwardActivationEventInfoReq stipulates:

‘Upon transmission of the token to machine nr. \(i\), machine nr. \(i + 1\) becomes white.’
**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:

\[ \text{ProbeNotSuccessful} \iff \text{hasToken(master)} \land \left( \text{tokenColor} = \text{black} \lor \text{color(master)} = \text{black} \right) \]

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

\[ \text{STARTPROBE} = \]

\[ \text{if} \ \text{ProbeNotSuccessful} \ \text{and} \ \text{self} = \text{master} \ \text{then} \]

\[ \text{COLOR\&FORWARDTOKEN} \]
**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 \( m \in \text{Machine} \)

\[
\begin{align*}
\text{mode}(m) & \in \{ \text{active, passive} \} \\
\text{color}(m) & \in \{ \text{black, white} \} \\
\text{hasToken}(\text{master}) & = \text{true} \\
\text{if } m \neq \text{master} \text{ then } \text{hasToken}(m) & = \text{false} \\
\text{Triggered}(m) & = \text{false} \\
\text{tokenColor} & = \text{black}
\end{align*}
\]

-- *InitializationReq*  
-- any initial machine coloring  
-- the master hasToken  
-- ... and nobody else  
-- meaning: NoMsgsAround  
-- triggers master to start a probe
The final concurrent ASM

\textbf{DistrTerminationDetect} =
\[(m, \text{TerminationDetection}_m)_{m \in \text{Machine}}\]

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

\textbf{TerminationDetection} =
\begin{align*}
\text{StartProbe} & \quad \text{-- done by master if \textit{ProbeNotSuccessful}} \\
\text{TriggerActivation} & \quad \text{-- spontaneously if active without token} \\
\text{BecomeActive} & \quad \text{-- if hasMessage} \\
\text{BecomePassive} & \quad \text{-- spontaneously if not hasMessage} \\
\text{PassToken} & \quad \text{-- if passive and \neq master}
\end{align*}
**TerminationDetected** iff

\[\text{hasToken}(\text{master}) \quad \text{and} \quad \text{mode}(\text{master}) = \text{passive} \]

\[\quad \text{and} \quad \text{color}(\text{master}) = \text{white} \quad \text{and} \quad \text{tokenColor} = \text{white}\]

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
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*

- E. Börger and A. Raschke: Modeling Companion for Software Practitioners. Springer 2018
  - http://modelingbook.informatik.uni-ulm.de
It is permitted to (re-) use these slides under the CC-BY-NC-SA licence

https://creativecommons.org/licenses/by-nc-sa/4.0/

i.e. in particular under the condition that
- the two original authors are mentioned
- modified slides are made available under the same licence
- the (re-) use is not commercial