Development of a
Network Topology Discovery Algorithm

Thai Son Hoang
David Basin
Hironobu Kuruma
Jean-Raymond Abrial
Thai Son Hoang, Hironobu Kuruma, David Basin, Jean-Raymond Abrial.

Developing Topology Discovery in Event-B

Science of Computer Programming (2009), 74 (11-12).

What is presented here is another development of the same problem
Why it is an Interesting Subject

- The final distributed algorithm is rather simple

- But the corresponding (formal) development is not that simple

- This example proposes some interesting techniques that can be used elsewhere

- We shall also insist on the importance of requirement analysis (requirements: what for?)
- Prologue
- Requirements
- Difficulties and Strategy
- Formal Development
- Conclusion
The **Master** and **Dog** paradigm
The Master rides a bike
The Dog tries to get to his Master
Master m, Dog d, and Goal g
Master and Dog
Master and Dog
Master and Dog: the Master Stops
- The Dog builds a local mental image of his Master’s position.

- The Dog reaches the Master if he stops for a sufficiently long time.
Another Dog, Just for the Fun ;-)
Requirements
- Requirements are very important

- Usually they are missing or poorly defined

- Requirements are informal statements

- They define the correctness criteria of a final implementation
- In our case, the master is a network

- This network may evolve as time goes

| We are given a finite set of nodes connected by oriented links which can be added or removed as time goes, thus forming a dynamic graph | REQ-1 |
The Network
The Network
The Network
The Network
The Network
The Network
- In our case, we have several dogs.

- Each node in the network is a dog.

- Each dog follow the master by forming a local image of the network.

- A node builds the network image by two different approaches (next slide)

- A dog reaches the master when its local image is the same as the network.
### Requirements 2: the Dogs

<table>
<thead>
<tr>
<th>REQ-2</th>
<th>When a link from node ( a ) to node ( b ) is added to or removed from the network then node ( b ) is DIRECTLY made aware of it</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-3</td>
<td>The neighbors of a node ( b ) are the nodes that are connected to ( b ) by a link ( l ) entering in ( b ).</td>
</tr>
<tr>
<td>REQ-4</td>
<td>Neighbors of ( b ) send to ( b ) their local networks with MESSAGES sent to it through the connecting link</td>
</tr>
</tbody>
</table>


The Network
The Network: Node Q is Made Aware of New Connection PQ
The Network: Node R is Made Aware of New Connection QR
The Network: Local Connection PQ in Q is sent from Q to R
The Network: Node S is Made Aware of New Connection RS
The Network: Local Connection QR in R is sent from R to S
The Network
The Network
The Network
- We have not shown any link removal.

- Messages from one node to the other travelled instantaneously.

- As a consequence, we had no loss of messages.

- But unfortunately:
  - links can be removed
  - messages do not travel instantaneously

- Hence messages between nodes can be lost.
<table>
<thead>
<tr>
<th>Requirement 3: Loss of Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>A message sent from (a) to (b) is lost if the link from (a) to (b), which existed when the message was sent, is broken before the message reaches (b)</td>
</tr>
</tbody>
</table>
A Message $m$ to be Sent from Node Q to Node P
The Message is Travelling
The Link is broken: the Message is Lost
The Link is not broken: the Message Reaches its Destination
- The dog-nodes can all reach together the master-network.

(under certain conditions)

| We must prove that under certain conditions the images of the graph built by nodes are all identical and equal to the graph itself | REQ-6 |

- The conditions are the following:
  - the network is not modified for a certain time,
  - the network is strongly connected when not modified
We are given a finite set of nodes connected by oriented links which can be added or removed as time goes, thus forming a dynamic graph.

<table>
<thead>
<tr>
<th>REQ-1</th>
</tr>
</thead>
</table>

When a link from node $a$ to node $b$ is added to or removed from the network then node $b$ is DIRECTLY made aware of it.

<table>
<thead>
<tr>
<th>REQ-2</th>
</tr>
</thead>
</table>

The neighbors of a node $b$ are the nodes that are connected to $b$ by a link $l$ entering in $b$.

<table>
<thead>
<tr>
<th>REQ-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neighbors of $b$ send to $b$ their local networks with MESSAGES sent to it through the connecting link</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>A message sent from $a$ to $b$ is lost if the link from $a$ to $b$, which existed when the message was sent, is broken before the message reaches $b$</td>
</tr>
<tr>
<td>We must prove that under certain conditions the images of the graph built by nodes are all identical and equal to the graph itself</td>
</tr>
</tbody>
</table>
Difficulties and Strategy
Modeling Difficulties

- Two ways of detecting link modifications: direct or with messages.

- Hence contradictory messages may circulate on the network.

- Which one reflects the real situation of the physical graph?

- Messages can be lost.

- What is the final condition (where all nodes know the graph)?

- How to express it?

- Where and how to start the modeling?
- Use abstraction

- Proceed by successive refinements

- Make proofs at each refinement level
Design Strategy: a Proposal

- Initial Model: Introducing the physical network.

- Refinement 1: Introducing a global logical network (detected).

- Refinement 2: Introducing local networks and message reservoirs.

- Refinement 3: Introducing ages of network changes.

- Refinement 4: Merging events.

- Refinement 5: Removing message reservoirs.

- Refinement 6: Finding the limit condition.

- Refinement 7: Handling loss of messages.
Formal Development
We are given a finite set of nodes connected by oriented links which can be added or removed as time goes, thus forming a dynamic graph.

- The physical network, \( NET \), is represented by a set of links \( L \).
Initial Model: External Events Adding or Removing Links

\[
\text{init} \quad NET := \emptyset
\]

Modify\_up
\[
\begin{align*}
\text{any} & \quad l \\
\text{where} & \quad l \not\in NET \\
\text{then} & \quad NET := NET \cup \{l\} \\
\text{end}
\end{align*}
\]

Modify\_dn
\[
\begin{align*}
\text{any} & \quad l \\
\text{where} & \quad l \in NET \\
\text{then} & \quad NET := NET \setminus \{l\} \\
\text{end}
\end{align*}
\]

- These events denote changes of the physical network \( NET \)
When a link from node $a$ to node $b$ is added to or removed from the graph then node $b$ is directly made aware of it

**variables:** $NET$  $net$

**inv1_1:** $net \subseteq L$

- We do not introduce the nodes yet (nor the local networks).

- The variable $net$ denotes the global logical image directly detected.

- It is an abstraction representing the direct detection of the network.
First Refinement: Initialisation

\[
\text{init} \\
NET := \emptyset \\
net := \emptyset
\]

- Events modify\_up and modify\_dn are not modified.
First Refinement: Updating the Logical Network (new events)

\[\begin{align*}
\text{discover\_up} & \quad \text{status} \\
& \quad \text{convergent} \\
& \quad \text{any} \\
& \quad l \\
& \quad \text{where} \\
& \quad l \in NET \setminus net \\
& \quad \text{then} \\
& \quad net := net \cup \{l\} \\
& \quad \text{end}
\end{align*}\]

\[\begin{align*}
\text{discover\_dn} & \quad \text{status} \\
& \quad \text{convergent} \\
& \quad \text{any} \\
& \quad l \\
& \quad \text{where} \\
& \quad l \in net \setminus NET \\
& \quad \text{then} \\
& \quad net := net \setminus \{l\} \\
& \quad \text{end}
\end{align*}\]

variant1: \((NET \setminus net) \cup (net \setminus NET)\)

- Mind the convergence (Master and Dog)
Second Refinement: Introducing Local Networks

sets: \( N \)

\[ \text{axm2}_1: \ finite(N) \]

variables: \( NET \quad net \quad l\_net \)

\[ \text{inv2}_1: \ l\_net \in N \leftrightarrow L \]

- \( l\_net \) denotes the local relation between nodes and links

- \( l\_net[\{n\}] \) is the set of links recorded in node \( n \)

- The local networks are modified in two distinct ways:
  - either directly when detected in the physical network
  - or indirectly by neighbors
Second Refinement: Introducing **Message Reservoirs**

variables:  
\[
\begin{align*}
m_{net\_up} & \\
m_{net\_dn} & 
\end{align*}
\]

<table>
<thead>
<tr>
<th>inv2.2:</th>
<th>(m_{net_up} \in N \leftrightarrow L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv2.3:</td>
<td>(m_{net_dn} \in N \leftrightarrow L)</td>
</tr>
</tbody>
</table>

| inv2.4: | \(m_{net\_up} \cap m_{net\_dn} = \emptyset\) |

- \(n \mapsto l \in m_{net\_up}\) means there is a message for node \(n\) about up-link \(l\).

- \(n \mapsto l \in m_{net\_dn}\) means there is a message for node \(n\) about dn-link \(l\).

- \(m_{net\_up}\) and \(m_{net\_dn}\) are abstractions with messages from neighbors

- Such messages are supposed to be in transit in \(m_{net\_up}\) and \(m_{net\_dn}\)
Second Refinement: Handling Message Reservoirs

discover_up
discover_dn

change_link_dn
change_link_up

m_net_dn
m_net_up
init

\[
\begin{align*}
NET & := \emptyset \\
net & := \emptyset \\
l_{net} & := \emptyset \\
m_{net\_up} & := \emptyset \\
m_{net\_dn} & := \emptyset
\end{align*}
\]
Second Refinement: Refining Event \texttt{discover\_up}

\begin{center}
\begin{tabular}{|l|}
\hline
\texttt{discover\_up} \\
\texttt{any} \\
\texttt{l} \\
\texttt{n} \\
\texttt{where} \\
\texttt{l} \in \texttt{NET} \setminus \texttt{net} \\
\texttt{n} \in \texttt{N} \\
\texttt{then} \\
\texttt{net} := \texttt{net} \cup \{l\} \\
\texttt{l\_net} := \texttt{l\_net} \cup \{n \mapsto l\} \\
\texttt{m\_net\_up} := \texttt{m\_net\_up} \cup ((\texttt{N} \setminus \{n\}) \times \{l\}) \\
\texttt{m\_net\_dn} := \texttt{m\_net\_dn} \setminus (\texttt{N} \times \{l\}) \\
\texttt{end} \\
\hline
\end{tabular}
\end{center}

- Node $n$, concerned by the change discovery, is chosen arbitrarily

- Local network of node $n$ is updated directly ($l\_net := \ldots$)

- New messages about link $l$ are prepared for nodes ($m\_net\_up := \ldots$)

- Old messages about link $l$ are MAGICALLY discarded ($m\_net\_dn := \ldots$)
Second Refinement: Refining Event `discover_dn`

```
discover_dn
    any
    l
    n
  where
    l ∈ net \ NET
    n ∈ N
  then
    net := net \ {l}
    l_net := l_net \ {n ↦ l}
    m_net_dn := m_net_dn ∪ ((N \ {n}) × {l})
    m_net_up := m_net_up \ (N × {l})
  end
```

- Node \( n \), concerned by the change discovery, is chosen arbitrarily
- Local network of node \( n \) is updated directly (\( l_net := \ldots \))
- New messages about link \( l \) are prepared for nodes (\( m_net_dn := \ldots \))
- Old messages about link \( l \) are MAGICALLY discarded (\( m_net_up := \ldots \))
change_link_up

status

anticipated

any

n

l

where

n \mapsto l \in m_{\text{net\_up}}

then

l_{\text{net}} := l_{\text{net}} \cup \{n \mapsto l\}

m_{\text{net\_up}} := m_{\text{net\_up}} \setminus \{n \mapsto l\}

end

- anticipated means the convergence is proved later.
Second Refinement: Updating \( l_{\text{net}} \) from "neighbors" (2)

\[
\begin{align*}
\text{change\_link\_dn} \\
\text{status} \\
\quad \text{anticipated} \\
\quad \text{any} \\
\quad \quad n \\
\quad \quad l \\
\text{where} \\
\quad n \mapsto l \in m_{\text{net\_dn}} \\
\text{then} \\
\quad l_{\text{net}} := l_{\text{net}} \setminus \{n \mapsto l\} \\
\quad m_{\text{net\_dn}} := m_{\text{net\_dn}} \setminus \{n \mapsto l\} \\
\text{end}
\end{align*}
\]
Second Refinement: Updating \texttt{l\_net} from "neighbors" (3)

\begin{center}
\begin{verbatim}
change\_link\_2
  status
    anticipated
    any
    ln
  where
    ln \in N \leftrightarrow L
  then
    l\_net := ln
end
\end{verbatim}
\end{center}

- To be explained in next refinement.
Third Refinement: Introducing ages

- Each modification is "decorated" with a natural number: the age.

- An even number for an addition.

- An odd number for a removal.

- Such numbers are always increasing.

- Consequence, removal of:
  - net
  - $m_{net\_up}$ and $m_{net\_dn}$
  - $l_{net}$
constants: \( \text{parity} \)

\begin{align*}
\text{axm3}_1: & \quad \text{parity} \in \mathbb{N} \rightarrow \{0, 1\} \\
\text{axm3}_2: & \quad \text{parity}(0) = 1 \\
\text{axm3}_3: & \quad \forall x \cdot \text{parity}(x + 1) = 1 - \text{parity}(x)
\end{align*}
Third Refinement: The *age* of a Link

variables: \[ \text{NET} \quad \text{age} \]

inv3\_1: \[ \text{age} \in L \rightarrow \mathbb{N} \]

- The age is recorded when events `discover_up` or `discover_dn` occur.

inv3\_2: \[ \forall l \cdot l \in \text{net} \iff \text{parity}(\text{age}(l)) = 1 \]

- Thanks to inv3\_2, variable `net` can be removed (replaced by `age`).
Third Refinement: Locally Recorded Age (1)

variables: \( NET \quad age \quad l\_age \)

\textbf{inv3\_3:} \( l\_age \in N \times L \rightarrow \mathbb{N} \)

- Each local network contains the age (when recorded) of each link

- This locally recorded age is at most equal to the "real" age (\textbf{inv3\_4})

\textbf{inv3\_4:} \( \forall n, l \cdot l\_age(n \mapsto l) \leq age(l) \)
inv3₅: \( \forall n, l \cdot n \mapsto l \in l_{\text{net}} \iff \text{parity}(l_{\text{age}}(n \mapsto l) = 1) \)

- Thanks to inv3₅, variable \( l_{\text{net}} \) can be removed (replaced by \( l_{\text{age}} \)).
Third Refinement: Messages with Ages (1)

variables: \[ \ldots m_{\text{net}} \]

inv3_6: \[ m_{\text{net}} \in N \times L \leftrightarrow \mathbb{N} \]

- Each message (when prepared) contains the age of the link
- Messages with the same link and different ages might exist

inv3_7: \[ \forall n, l, a \cdot n \mapsto l \mapsto a \in m_{\text{net}} \Rightarrow a \leq \text{age}(l) \]

- The recorded age in a message is at most equal to the real age

inv3_8: \[ \forall n, l, a \cdot \text{l_age}(n \mapsto l) < a \land a \leq \text{age}(l) \Rightarrow n \mapsto l \mapsto a \in m_{\text{net}} \]

- Invariant \textbf{inv3_8} will be used in refinement 5
Third Refinement: Messages with Ages (2)

inv3.9: \( \forall n, l \cdot n \mapsto l \in m_{\text{net_up}} \iff n \mapsto l \mapsto \text{age}(l) \in m_{\text{net}} \land \text{parity}(\text{age}(l)) = 1 \)

inv3.10: \( \forall n, l \cdot n \mapsto l \in m_{\text{net_dn}} \iff n \mapsto l \mapsto \text{age}(l) \in m_{\text{net}} \land \text{parity}(\text{age}(l)) = 0 \)

- \( m_{\text{net_up}} \) and \( m_{\text{net_dn}} \) can be removed (replaced by \( m_{\text{net}} \)).
init

\[ NET := \emptyset \]
\[ age := L \times \{0\} \]
\[ l\_age := N \times L \times \{0\} \]
\[ m\_net := \emptyset \]
(abstract.) discover_up

any
l
n
where
l ∈ NET
l /∈ net
n ∈ N
then
net := net ∪ {l}
l_net := l_net ∪ \{n \mapsto l\}
m_net_up := m_net_up ∪ ((N \setminus \{n\}) × \{l\})
m_net_dn := m_net_dn \setminus (N × \{l\})
end

discover_up

any
l
n
where
l ∈ NET
parity(age(l)) = 0 /\ l /∈ net ⇔ parity(age(l)) = 0 (inv3.2) */
n ∈ N
then
\begin{align*}
\text{age}(l) & := \text{age}(l) + 1 \\
\text{l_age}(n \mapsto l) & := \text{age}(l) + 1 \\
\text{m_net} & := \text{m_net} ∪ ((N \setminus \{n\}) \times \{l\} \times \{\text{age}(l) + 1\})
\end{align*}
end
Third Refinement: Events (2)

(abstract-)discover\_dn

\[
\begin{align*}
\text{any} & \quad l \\
\text{any} & \quad n \\
\text{where} & \quad l \notin \text{NET} \\
& \quad l \in \text{net} \\
& \quad n \in \mathbb{N} \\
\text{then} & \quad \text{net} := \text{net} \setminus \{l\} \\
& \quad l\_net := l\_net \setminus \{n \mapsto l\} \\
& \quad m\_net\_dn := m\_net\_dn \cup ((\mathbb{N} \setminus \{n\}) \times \{l\}) \\
& \quad m\_net\_up := m\_net\_up \setminus (\mathbb{N} \times \{l\}) \\
\end{align*}
\]

end

discover\_dn

\[
\begin{align*}
\text{any} & \quad l \\
\text{any} & \quad n \\
\text{where} & \quad l \notin \text{NET} \\
& \quad \text{parity}(\text{age}(l)) = 1 \quad /\!\!/ \quad l \in \text{net} \Leftrightarrow \text{parity}(\text{age}(l)) = 1 \quad \text{(inv3.2) */} \\
& \quad n \in \mathbb{N} \\
\text{then} & \quad \text{age}(l) := \text{age}(l) + 1 \\
& \quad l\_age(n \mapsto l) := \text{age}(l) + 1 \\
& \quad m\_net := m\_net \cup ((\mathbb{N} \setminus \{n\}) \times \{l\} \times \{\text{age}(l) + 1\})) \\
\end{align*}
\]
end
Third Refinement: Events (3)

- Events **discover_up** and **discover_dn** will be merged (same actions).

- The merged event will be convergent
### Third Refinement: Events (4)

(abstract-)change link up

```latex
\textbf{status} \quad \text{anticipated}
\textbf{any} \quad n \quad l
\textbf{where} \quad n \mapsto l \in m_{\text{net}_\text{up}}
\textbf{then}
\quad l_{\text{net}} := l_{\text{net}} \cup \{n \mapsto l\}
\quad m_{\text{net}_\text{up}} := m_{\text{net}_\text{up}} \setminus \{n \mapsto l\}
\textbf{end}
```

(abstract-)change link down

```latex
\textbf{status} \quad \text{anticipated}
\textbf{any} \quad n \quad l
\textbf{where} \quad n \mapsto l \in m_{\text{net}_\text{dn}}
\textbf{then}
\quad l_{\text{net}} := l_{\text{net}} \setminus \{n \mapsto l\}
\quad m_{\text{net}_\text{dn}} := m_{\text{net}_\text{dn}} \setminus \{n \mapsto l\}
\textbf{end}
```

change link up

```latex
\textbf{status} \quad \text{convergent}
\textbf{any} \quad n \quad l \quad x
\textbf{where} \quad x = \text{age}(l)
\quad n \mapsto l \mapsto x \in m_{\text{net}} \quad /\!* \text{inv3.9} */\!
\quad \text{parity}(x) = 1
\textbf{then}
\quad l_{\text{age}}(n \mapsto l) := x
\quad m_{\text{net}} := m_{\text{net}} \setminus \{n \mapsto l \mapsto x\}
\textbf{end}
```

change link down

```latex
\textbf{status} \quad \text{convergent}
\textbf{any} \quad n \quad l \quad x
\textbf{where} \quad x = \text{age}(l)
\quad n \mapsto l \mapsto x \in m_{\text{net}} \quad /\!* \text{inv3.10} */\!
\quad \text{parity}(x) = 0
\textbf{then}
\quad l_{\text{age}}(n \mapsto l) := x
\quad m_{\text{net}} := m_{\text{net}} \setminus \{n \mapsto l \mapsto x\}
\textbf{end}
```
- The age $x$ in the message is different from $age(l)$ but node $n$ cannot know that
- It can only observe that it is greater than $l_age(n \mapsto l)$

change_link_2

status
convergent

any
$n$
$l$
$x$

where
$x \neq age(l)$
$n \mapsto l \mapsto x \in m_net$
$x > l_age(n \mapsto l)$

with
$(parity(x) = 0 \Rightarrow ln = l_net \setminus \{n \mapsto l\}) \land$
$(parity(x) = 1 \Rightarrow ln = l_net \cup \{n \mapsto l\})$

then
$l_age(n \mapsto l) := x$
$m_net := m_net \setminus \{n \mapsto l \mapsto x\}$

end

inv3_13: finite($m_net$)

variant3: $m_net$

- Events change_link_up, change_link_dn and change_link_2 will be merged (same actions).
- The merged event will be convergent too.
- These events will be **merged** (same actions)

- But the **merged event will not be convergent**
  (because event discard_2 is not convergent)
discover
refines
discover_up
discover_dn
any
l
n
where
\( l \in NET \Leftrightarrow parity(age(l)) = 0 \)
\( n \in N \)
then
\( age(l) := age(l) + 1 \)
\( m_{net} := m_{net} \cup ((N \setminus \{n\}) \times \{l\} \times \{age(l) + 1\}) \)
\( l\_age(n \mapsto l) := age(l) + 1 \)
end

- The guard does not involve the variable \( m_{net} \).
change_link
   refines
       change_link_up
       change_link_dn
       change_link_2
any
   n
   l
   x
where
   n ↦ l ↦ x ∈ m_net
   x > l_age(n ↦ l)
thenunder
   l_age(n ↦ l) := x
   m_net := m_net \ {n ↦ l ↦ x}
end

- In the next refinement, the guard will be made independent from \( m_net \).
discard
  refines
discard_1
discard_2
any
  n
  l
  x
where
  x \leq \text{age}(n \mapsto l)
then
  m_{net} := m_{net} \setminus \{n \mapsto l \mapsto x\}
end

- The guard does not involve the variable \textit{m_{net}}.
constants:  \( \text{fst} \quad \text{snd} \quad \text{link} \)

\begin{align*}
\text{axm5.1:} & \quad \text{fst} \in L \rightarrow N \\
\text{axm5.2:} & \quad \text{snd} \in L \rightarrow N \\
\text{axm5.3:} & \quad \text{link} \in N \times N \rightarrow L \\
\text{axm5.4:} & \quad \forall n, m \cdot \text{fst}(\text{link}(n \mapsto m)) = n \\
\text{axm5.5:} & \quad \forall n, m \cdot \text{snd}(\text{link}(n \mapsto m)) = m \\
\text{axm5.6:} & \quad \forall l \cdot \text{link} (\text{fst}(l) \mapsto \text{snd}(l)) = l
\end{align*}
Fifth Refinement: Removing variable $m_{\text{net}}$ (1)

```plaintext
change_link
any
  l
  x
  k
where
  k \in NET
  l\_age(snd(k) \mapsto l) < x
  x \leq l\_age(fst(k) \mapsto l)      /* n \mapsto l \mapsto x \in m_{\text{net}} */
with
  n = snd(k)
then
  l\_age(snd(k) \mapsto l) := x
  /* m_{\text{net}} := m_{\text{net}} \setminus \{snd(k) \mapsto l \mapsto x\} */
end
```

- The proof of guard strengthening uses invariant $\text{inv3\_8}$

\[ \forall n, l, x \cdot l\_age(n \mapsto l) < x \land x \leq age(l) \Rightarrow n \mapsto l \mapsto x \in m_{\text{net}} \]

- and invariant $\text{inv3\_4}$

\[ \forall n, l \cdot l\_age(n \mapsto l) \leq age(l) \]
Fifth Refinement: Removing variable $m_{\text{net}}$ (2)

- $m_{\text{net}}$ is removed since it does not appear in any guard

```
discover
  any
  l
where
  l ∈ NET ⇔ parity(age(l)) = 0
with
  n = snd(l)
then
  age(l) := age(l) + 1
  l_age(snd(l) ↦ l) := age(l) + 1
end
```
- `$m_{net}$` is removed since it does not appear in any guard

```
discard
  any
    l
    x
    k
  where
    k ∈ NET
    x ≤ \text{age}(\text{snd}(k) \mapsto l)
  with
    n = \text{snd}(k)
  then
    skip
end
```
Sixth Refinement: Defining the Graph $G$

**variables:** $age$ $l_{age}$ $G$

**inv6\_1:** $G \in N \leftrightarrow N$

**inv6\_2:** $\forall l \cdot l \in NET \leftrightarrow fst(l) \mapsto snd(l) \in G$

**inv6\_3:** $\forall l \cdot age(l) = l_{age}(snd(l) \mapsto l)$

- Invariant **inv6\_2** allows us to remove the variable $NET$.

- Invariant **inv6\_3** will allow us to remove the variable $age$ (in next refinement)
Sixth Refinement: Initialisation

\[
\text{init} \\
G \quad := \quad \emptyset \\
age \quad := \quad L \times \{0\} \\
l\_age \quad := \quad N \times L \times \{0\}
\]
discover any $l$

where $\text{fst}(l) \mapsto \text{snd}(l) \in G \iff \text{parity}(\text{age}(l)) = 0$

then

\[
\text{age}(l) := \text{age}(l) + 1
\]

\[
\text{l_age}(\text{snd}(l) \mapsto l) := \text{age}(l) + 1
\]

end
Neighbors of $b$ send to $b$ their local networks with MESSAGES sent to it through the connecting link

change_link

any
l
x
k

where

\[ \text{fst}(k) \mapsto \text{snd}(k) \in G \]
\[ x > \text{age}(\text{snd}(k) \mapsto l) \]
\[ x \leq \text{age}(\text{fst}(k) \mapsto l) \]

/* $x$ is explained in next refinement */

with

\[ n = \text{snd}(k) \]

then

\[ \text{age}(\text{snd}(k) \mapsto l) := x \]

end

- Node $\text{fst}(k)$ is a neighbor of $\text{snd}(k)$ since $\text{fst}(k) \mapsto \text{snd}(k) \in G$
Sixth Refinement: Finding the Limit Condition (1)

We must prove that under certain conditions the images of the graph built by nodes are all identical and equal to the graph itself

- To be proved

Convergent events discover and change link deadlocks
Physical graph is strongly connected
⇒
Local networks are all equal to physical network

- There is nothing to discover (the graph is still)
- There is nothing significative to transmitt.
- Note that these events are both convergent.
- Guard of the event **discover**:

\[ \exists l \cdot \text{fst}(l) \leftrightarrow \text{snd}(l) \in G \iff \text{parity}(\text{age}(l)) = 0 \]

- Negation of the guard of the event **discover** (deadlock condition):

\[ \neg \exists l \cdot \text{fst}(l) \leftrightarrow \text{snd}(l) \in G \iff \text{parity}(\text{age}(l)) = 0 \]
\[ \iff \forall l \cdot \text{fst}(l) \leftrightarrow \text{snd}(l) \in G \iff \text{parity}(\text{age}(l)) = 1 \]
\[ \iff \forall l \cdot l \in NET \iff \text{parity}(\text{age}(l)) = 1 \]
- Guard of the event \texttt{change\_link}:

\[
\exists l, x, k \cdot \text{fst}(k) \rightarrow \text{snd}(k) \in G \land
\]
\[
x > \text{age}(\text{snd}(k) \rightarrow l) \land
\]
\[
x \leq \text{age}(\text{fst}(k) \rightarrow l)
\]

- Negation of the guard of the event \texttt{change\_link} (deadlock condition):

\[
\neg \exists l, x, k \cdot \text{fst}(k) \rightarrow \text{snd}(k) \in G \land
\]
\[
x > \text{age}(\text{snd}(k) \rightarrow l) \land
\]
\[
x \leq \text{age}(\text{fst}(k) \rightarrow l)
\]
\[
\iff
\]
\[
\forall l, x, k \cdot \text{fst}(k) \rightarrow \text{snd}(k) \in G
\]
\[
\Rightarrow
\]
\[
x \leq \text{age}(\text{snd}(k) \rightarrow l) \lor x > \text{age}(\text{fst}(k) \rightarrow l)
\]
\[
\iff
\]
\[
\forall l, x, a, b \cdot a \leftrightarrow b \in G \Rightarrow x \leq \text{age}(b \leftrightarrow l) \lor x > \text{age}(a \leftrightarrow l)
\]
\[
\iff
\]
\[
\forall l, a, b \cdot a \leftrightarrow b \in G \Rightarrow \text{age}(a \leftrightarrow l) \leq \text{age}(b \leftrightarrow l)
\]
Sixth Refinement: Finding the Limit Condition (4)

- Observe strong connectivity of graph $G$

\[
\text{thm6}_1: \quad \forall l, a, b \cdot a \leftrightarrow b \in G \Rightarrow \text{l\_age}(a \leftrightarrow l) \leq \text{l\_age}(b \leftrightarrow l) \\
\forall s \cdot s \neq \emptyset \land G[s] \subseteq s \Rightarrow N \subseteq s \\
\Rightarrow \\
\forall l, n \cdot \text{l\_age}(n \leftrightarrow l) = \text{age}(l)
\]

- Hint: instantiate $s$ with $\{ n \mid \text{l\_age}(n \leftrightarrow l) = \text{age}(l) \}$.

\[
\text{thm6}_2: \quad \forall l \cdot l \in NET \iff \text{parity}(\text{age}(l)) = 1 \\
\forall l, a, b \cdot a \leftrightarrow b \in G \Rightarrow \text{l\_age}(a \leftrightarrow l) \leq \text{l\_age}(b \leftrightarrow l) \\
\forall s \cdot s \neq \emptyset \land G[s] \subseteq s \Rightarrow N \subseteq s \\
\Rightarrow \\
\forall l, n \cdot l \in NET \iff \text{parity}(\text{l\_age}(n \leftrightarrow l)) = 1
\]
Sixth Refinement: Finding the Limit Condition (5)

**thm6.2:** \( \forall l \cdot l \in NET \Leftrightarrow \text{parity}(\text{age}(l)) = 1 \)

\( \forall l, a, b \cdot a \mapsto b \in G \Rightarrow \text{age}(a \mapsto l) \leq \text{age}(b \mapsto l) \)

\( \forall s \cdot s \neq \emptyset \land G[s] \subseteq s \Rightarrow N \subseteq s \)

\( \Rightarrow \)

\( \forall l, n \cdot l \in NET \Leftrightarrow \text{parity}(\text{age}(n \mapsto l)) = 1 \)

**thm6.3:** \( \forall l \cdot l \in NET \Leftrightarrow \text{parity}(\text{age}(l)) = 1 \)

\( \forall l, a, b \cdot a \mapsto b \in G \Rightarrow \text{age}(a \mapsto l) \leq \text{age}(b \mapsto l) \)

\( \forall s \cdot s \neq \emptyset \land G[s] \subseteq s \Rightarrow N \subseteq s \)

\( \Rightarrow \)

\( \forall n \cdot l\_\text{net}[\{n\}] = NET \)

- Hint: Use invariant **inv3.5**

**inv3.5:** \( \forall n, l \cdot l \in l\_\text{net}[\{n\}] \Leftrightarrow \text{parity}(\text{age}(n \mapsto l)) = 1 \)
**thm6.3:**

\[
\forall l \cdot l \in NET \iff \text{parity}(\text{age}(l)) = 1 \\
\forall l, a, b \cdot a \leftrightarrow b \in G \Rightarrow l_{\text{age}}(a \leftrightarrow l) \leq l_{\text{age}}(b \leftrightarrow l) \\
\forall s \cdot s \neq \emptyset \land G[s] \subseteq s \Rightarrow N \subseteq s \\
\Rightarrow \\
\forall n \cdot \text{l.net}[\{n\}] = NET
\]

Event discover deadlocks
Event change_link deadlocks

Physical graph is strongly connected

⇒

Local networks are all equal to physical network
Modify up
any
l
where
\( \text{fst}(l) \mapsto \text{snd}(l) \notin G \)
then
\( G := G \cup \{ \text{fst}(l) \mapsto \text{snd}(l) \} \)
end

Modify dn
any
l
where
\( \text{fst}(l) \mapsto \text{snd}(l) \in G \)
then
\( G := G \setminus \{ \text{fst}(l) \mapsto \text{snd}(l) \} \)
end
discard

any

l

x

k

where

\( \text{fst}(k) \mapsto \text{snd}(k) \in G \)

\( x \leq \text{age}(\text{snd}(k) \mapsto l) \)

then

skip

derase
Seventh Refinement: Introducing Message Channels \( m \)

**variables:** \( G \), \( l_{age} \), \( m \)

**inv7_1:** \( m \subseteq L \times L \times \mathbb{N} \)

**inv7_2:** \( \forall k, l, x \cdot k \mapsto l \mapsto x \in m \Rightarrow x \leq l_{age}(fst(k) \mapsto l) \)

**inv7_3:** \( \forall k, l, x \cdot k \mapsto l \mapsto x \in m \Rightarrow fst(k) \mapsto snd(k) \in G \)

- \( k \mapsto l \mapsto x \in m \) means a message dealing with link \( l \) and age \( x \) is travelling on link \( k \).

- **inv7_2** says that ages at the origin of a message, \( l_{age}(fst(k) \mapsto l) \), can only increase while message is travelling.

- **inv7_3** says that a message is always travelling on an existing link.
init

\[ G := \emptyset \]
\[ m := \emptyset \]
\[ l\_age := N \times L \times \{0\} \]
A message sent from \( a \) to \( b \) is lost if the link from \( a \) to \( b \), which existed when the message was sent, is broken before the message reaches \( b \)

- All messages travelling on link \( l \) are lost when \( l \) is broken.
Seventh Refinement: Events (2)

(abstract-)discover

\[
\begin{align*}
\text{any} & \quad l \\
\text{where} & \quad \text{fst}(l) \mapsto \text{snd}(l) \in G \iff \text{parity}(\text{age}(l)) = 0 \\
\text{then} & \quad \text{age}(l) := \text{age}(l) + 1 \\
\text{end} \\
\text{end}
\end{align*}
\]

\text{inv6.3: } \forall l \cdot \text{age}(l) = \text{l_age(snd(l) }\mapsto l) 

\[
\begin{align*}
\text{discover} & \quad \text{any} \\
\text{any} & \quad l \\
\text{where} & \quad \text{fst}(l) \mapsto \text{snd}(l) \in G \iff \text{parity}(\text{l_age(snd(l) }\mapsto l)) = 0 \\
\text{then} & \quad \text{l_age(snd(l) }\mapsto l) := \text{l_age(snd(l) }\mapsto l) + 1 \\
\text{end}
\end{align*}
\]
Neighbors of $b$ send to $b$ their local networks with MESSAGES sent to it through the connecting link

send_message
  any
  $k$
  $l$
  where
  $\text{fst}(k) \mapsto \text{snd}(k) \in G$
  $l \in L$
  then
  $m := m \cup \{k \mapsto l \mapsto \text{age}(\text{fst}(k) \mapsto l)\}$
end

- Observe the non-determinacy on $l$
- This event is not convergent: a message can always be sent.
Seventh Refinement: Events (3)

(abstract-)change

any
l
x
k
where
\( \text{fst}(k) \mapsto \text{snd}(k) \in G \)
\( x > \text{age}(\text{snd}(k) \mapsto l) \)
\( x \leq \text{age}(\text{fst}(k) \mapsto l) \)
with
\( n = \text{snd}(k) \)
then
\( \text{age}(\text{snd}(k) \mapsto l) := x \)
end

inv7.2: \( \forall k, l, x \cdot k \mapsto l \mapsto x \in m \Rightarrow x \leq \text{age}(\text{fst}(k) \mapsto l) \)

inv7.3: \( \forall k, l, x \cdot k \mapsto l \mapsto x \in m \Rightarrow \text{fst}(k) \mapsto \text{snd}(k) \in G \)

(abstract-)discard

any
l
x
k
where
\( \text{fst}(k) \mapsto \text{snd}(k) \in G \)
\( x \leq \text{age}(\text{snd}(k) \mapsto l) \)
then
skip
end

accept_message
refines
change_link
any
l
x
k
where
\( k \mapsto l \mapsto x \in m \)
\( x > \text{age}(\text{snd}(k) \mapsto l) \)
then
\( \text{age}(\text{snd}(k) \mapsto l) := x \)
\( m := m \setminus \{ k \mapsto l \mapsto x \} \)
end

discard_message
refines
discard
any
l
x
k
where
\( k \mapsto l \mapsto x \in m \)
\( x \leq \text{age}(\text{snd}(k) \mapsto l) \)
then
\( m := m \setminus \{ k \mapsto l \mapsto x \} \)
end
- The final result is particularly simple:

  - Three variables:
    - $G, l_{age}, m$

  - Simple events:
    - Modify_up, Modify_dn
    - discover
    - send_message
    - accept_message
    - discard_message

- However, the development is not that simple
## Conclusion: Proof Statistics

<table>
<thead>
<tr>
<th>model</th>
<th>proofs</th>
<th>auto.</th>
<th>manual</th>
<th>% auto.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ref. 1</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Ref. 2</td>
<td>19</td>
<td>19</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Ref. 3</td>
<td>115</td>
<td>79</td>
<td>36</td>
<td>68</td>
</tr>
<tr>
<td>Ref. 4</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>83</td>
</tr>
<tr>
<td>Ref. 5</td>
<td>18</td>
<td>17</td>
<td>1</td>
<td>94</td>
</tr>
<tr>
<td>Ref. 6</td>
<td>13</td>
<td>10</td>
<td>3</td>
<td>76</td>
</tr>
<tr>
<td>Ref. 7</td>
<td>38</td>
<td>32</td>
<td>6</td>
<td>84</td>
</tr>
<tr>
<td>Total</td>
<td>212</td>
<td>165</td>
<td>47</td>
<td>77</td>
</tr>
</tbody>
</table>
THAT’S ALL