MODULE *AnyLamportClock*

TLA+ specification of *Lamport*'s distributed mutual-exclusion algorithm as mentioned in Assignment 2

EXTENDS *Naturals*, *Sequences*, *TLC*

The parameter N represents the number of processes. The parameter *maxClock* is used only for model checking in order to keep the state space finite.

CONSTANT N, *maxClock*

ASSUME $NType \triangleq N \in Nat$
ASSUME $maxClockType \triangleq maxClock \in Nat$

$Proc \triangleq 1 \,..\, N$
$Clock \triangleq Nat \setminus \{0\}$

For model checking, add *ClockConstraint* as a state constraint to ensure a finite state space and override the definition of *Clock* by $1 \,..\, maxClock + 1$ so that *TLC* can evaluate the definition of *Message*.

VARIABLES
- *clock*, — local clock of each process
- *req*, — requests received from processes stored as sequence (clock transmitted with request)
- *ack*, — acknowledgements received from processes
- *network*, — messages sent but not yet received
- *crit* — set of processes in critical section

Messages used in the algorithm.

$ReqMessage(c) \triangleq [type \mapsto \text{"req"}, clock \mapsto c]$
$AckMessage(c) \triangleq [type \mapsto \text{"ack"}, clock \mapsto c]$
$RelMessage \triangleq [type \mapsto \text{"rel"}, clock \mapsto 0]$

$Message \triangleq \{RelMessage\} \cup \{ReqMessage(c) : c \in Clock\} \cup \{AckMessage(c) : c \in Clock\}$

The type correctness predicate.

$TypeOK \triangleq$

clock[*p*] is the local clock of process *p*

$\land\ clock \in [Proc \rightarrow Clock]$

req[*p*][*q*] stores sequence of clock associated with request from *q* received by *p*, empty if none

$\land\ req \in [Proc \rightarrow [Proc \rightarrow Seq(Nat)]]$

ack[*p*] stores the processes that have *ack*'ed *p*'s request

$\land\ ack \in [Proc \rightarrow [Proc \rightarrow Nat]]$

network[*p*][*q*]: queue of messages from *p* to *q* – pairwise *FIFO* communication

$\land\ network \in [Proc \rightarrow [Proc \rightarrow Seq(Message)]]$

set of processes in critical section: should be empty or singleton

$\land\ crit \in \text{SUBSET } Proc$

The initial state predicate.

$$
\begin{array}{l}
\mathit{Init} \triangleq \\
\quad \land\ \mathit{clock} = [p \in \mathit{Proc} \mapsto 1] \\
\quad \land\ \mathit{req} = [p \in \mathit{Proc} \mapsto [q \in \mathit{Proc} \mapsto \langle\rangle]] \\
\quad \land\ \mathit{ack} = [p \in \mathit{Proc} \mapsto [q \in \mathit{Proc} \mapsto 0]] \\
\quad \land\ \mathit{network} = [p \in \mathit{Proc} \mapsto [q \in \mathit{Proc} \mapsto \langle\rangle]] \\
\quad \land\ \mathit{crit} = \{\}
\end{array}
$$

$\mathit{beats}(p,\ q)$ is true if process p believes that its request has higher priority than q's request. This is true if either p has not received a request from q or p's request has a smaller clock value than any received from q. If there is a tie, the numerical process ID decides.

$$
\begin{array}{l}
\mathit{beats}(p,\ q) \triangleq \\
\quad \lor\ \mathit{req}[p][q] = \langle\rangle \\
\quad \lor\ \land\ \mathit{req}[p][p] \neq \langle\rangle \\
\qquad \land\ \mathit{req}[p][q] \neq \langle\rangle \\
\qquad \land\ \textsc{let}\ \mathit{pTop} \triangleq \mathit{Head}(\mathit{req}[p][p]) \\
\qquad\qquad\qquad \mathit{qTop} \triangleq \mathit{Head}(\mathit{req}[p][q]) \\
\qquad\quad \textsc{in} \\
\qquad\qquad \lor\ \mathit{pTop} < \mathit{qTop} \\
\qquad\qquad \lor\ \mathit{pTop} = \mathit{qTop} \land p < q
\end{array}
$$

Broadcast a message: send it to all processes except the sender.

$$
\begin{array}{l}
\mathit{Broadcast}(s,\ m) \triangleq \\
\quad [r \in \mathit{Proc} \mapsto \textsc{if}\ s = r\ \textsc{then}\ \mathit{network}[s][r]\ \textsc{else}\ \mathit{Append}(\mathit{network}[s][r],\ m)]
\end{array}
$$

Process p requests access to critical section. Increment Clock time also

$$
\begin{array}{l}
\mathit{Request}(p) \triangleq \\
\quad \land\ \mathit{req}' = [\mathit{req}\ \textsc{except}\ ![p][p] = \mathit{Append}(@,\ \mathit{clock}[p] + 1)] \\
\quad \land\ \mathit{network}' = [\mathit{network}\ \textsc{except}\ ![p] = \mathit{Broadcast}(p,\ \mathit{ReqMessage}(\mathit{clock}[p] + 1))] \\
\quad \land\ \mathit{clock}' = [\mathit{clock}\ \textsc{except}\ ![p] = @ + 1] \\
\quad \land\ \textsc{unchanged}\ \langle \mathit{ack},\ \mathit{crit} \rangle
\end{array}
$$

Process p receives a request from q It adds this new request to its request Queue and increment the clock $\mathit{time.Also}$ it acknowledges it with a timestamp.

$$
\begin{array}{l}
\mathit{ReceiveRequest}(p,\ q) \triangleq \\
\quad \land\ \mathit{network}[q][p] \neq \langle\rangle \\
\qquad \land\ \textsc{let}\ m \triangleq \mathit{Head}(\mathit{network}[q][p]) \\
\qquad\quad \textsc{in}\quad \land\ m.\mathit{type} = \text{“req”} \\
\qquad\qquad\qquad \land\ \textsc{let}\ c \triangleq m.\mathit{clock} \\
\qquad\qquad\qquad\qquad\quad \mathit{nextClock} \triangleq \textsc{if}\ c < \mathit{clock}[p]\ \textsc{then}\ \mathit{clock}[p] + 1\ \textsc{else}\ c + 1 \\
\qquad\qquad\qquad\quad \textsc{in}\quad \land\ \mathit{req}' = [\mathit{req}\ \textsc{except}\ ![p][q] = \mathit{Append}(@,\ c)] \\
\qquad\qquad\qquad\qquad\quad \land\ \mathit{clock}' = [\mathit{clock}\ \textsc{except}\ ![p] = \textsc{if}\ c > \mathit{clock}[p]\ \textsc{then}\ c + 1\ \textsc{else}\ @ + 1] \\
\qquad\qquad\qquad\qquad\quad \land\ \mathit{network}' = [\mathit{network}\ \textsc{except}\ ![q][p] = \mathit{Tail}(@), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![p][q] = \mathit{Append}(@,\ \mathit{AckMessage}(\mathit{nextClock}))] \\
\qquad\qquad\qquad\qquad\quad \land\ \textsc{unchanged}\ \langle \mathit{ack},\ \mathit{crit} \rangle
\end{array}
$$

Process p receives an acknowledgement from q.

$ReceiveAck(p, q) \triangleq$
$\land\ network[q][p] \neq \langle\rangle$
$\quad \land \text{LET } m \triangleq Head(network[q][p])$
$\quad\quad \text{IN} \quad \land\ m.type = \text{"ack"}$
$\quad\quad\quad \land\ ack' = [ack \text{ EXCEPT } ![p][q] = m.clock]$
$\quad\quad\quad \land\ network' = [network \text{ EXCEPT } ![q][p] = Tail(@)]$
$\quad\quad\quad \land\ \text{UNCHANGED } \langle clock, req, crit\rangle$

Process p enters the critical *section.Process* enter *CS* if it has a request in the request queue and it beats all other processes. Also the timestamp of all the *Ack* should be greater than the request time stamp

$Enter(p) \triangleq$
$\land\ req[p][p] \neq \langle\rangle$
$\quad \land \text{LET } pTop \triangleq Head(req[p][p])$
$\quad\quad \text{IN} \quad \land\ \forall q \in Proc \setminus \{p\} : pTop < ack[p][q]$
$\quad\quad\quad \land\ \forall q \in Proc \setminus \{p\} : beats(p, q)$
$\quad\quad\quad \land\ crit' = crit \cup \{p\}$
$\quad\quad\quad \land\ \text{UNCHANGED } \langle clock, req, ack, network\rangle$

Process p exits the critical section and notifies other *processes.It* deletes any one of its request from its request queue

$Exit(p) \triangleq$
$\land \text{LET } l \triangleq RandomElement(1 \mathrel{..} (Len(req[p][p])))\text{IN}$
$\quad \land\ p \in crit$
$\quad \land\ crit' = crit \setminus \{p\}$
$\quad \land\ network' = [network \text{ EXCEPT } ![p] = Broadcast(p, RelMessage)]$
$\quad \land\ req' = [req \text{ EXCEPT } ![p][p] = Tail(@)]$ Uncomment this line of code to check for *Liveness*
$\quad \land\ req' = [req \text{ EXCEPT } ![p][p] = [i \in 1 \mathrel{..} (Len(@) - 1) \mapsto \text{IF } i < l \text{ THEN } @[i] \text{ ELSE } @[i+1]]]$ Comment th
$\quad \land\ \text{UNCHANGED } \langle clock, ack\rangle$

Process p receives a release notification from q and deletes any request from its request queue

$ReceiveRelease(p, q) \triangleq$
$\land \text{LET } l \triangleq RandomElement(1 \mathrel{..} (Len(req[p][q])))\text{IN}$
$\quad \land\ network[q][p] \neq \langle\rangle$
$\quad \land\ \text{LET } m \triangleq Head(network[q][p])$
$\quad\quad \text{IN} \quad \land\ m.type = \text{"rel"}$
$\quad\quad\quad \land\ req' = [req \text{ EXCEPT } ![p][q] = Tail(@)]$
$\quad\quad\quad \land\ req' = [req \text{ EXCEPT } ![p][q] = [i \in 1 \mathrel{..} (Len(@) - 1) \mapsto \text{IF } i < l \text{ THEN } @[i] \text{ ELSE } @[i+1]]]$
$\quad\quad\quad \land\ network' = [network \text{ EXCEPT } ![q][p] = Tail(@)]$
$\quad\quad\quad \land\ \text{UNCHANGED } \langle clock, ack, crit\rangle$

$\land Print(l, \text{TRUE})$

Next-state relation.

$Next \triangleq$
$\quad \lor \exists p \in Proc : Request(p) \lor Enter(p) \lor Exit(p)$
$\quad \lor \exists p \in Proc : \exists q \in Proc \setminus \{p\} : ReceiveRequest(p, q) \lor ReceiveAck(p, q) \lor ReceiveRelease(p, q)$

$vars \triangleq \langle req, network, clock, ack, crit \rangle$

$Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)$

A state constraint that is useful for validating the specification using finite-state model *checking.Constraints* imposed on clock,network and request

$$
\begin{aligned}
StateConstraint \triangleq\ & \land \forall p \in Proc : clock[p] \leq maxClock \\
& \land \forall p, q \in Proc : Len(network[p][q]) \leq 1 \\
& \land \forall p, q \in Proc : Len(req[p][q]) \leq 2
\end{aligned}
$$

The main safety property of mutual exclusion.

$Mutex \triangleq \forall p, q \in crit : p = q$

$Live \triangleq \forall p \in Proc : \Diamond(p \in crit)$

---

* Modification History
* Last modified Sun *Oct* 07 20:57:29 *EDT* 2018 by *mehuljain*
* Created Sun *Oct* 07 15:39:52 *EDT* 2018 by *mehuljain*