MODULE *Spec_3*

EXTENDS *Naturals*, *Sequences*
CONSTANT N, *maxClock*

N nodes execute the algorithm. Each node is represented by two processes: the main "site" that requests access to the critical section, and a "communicator" that asynchronously receives messages directed to the "site" and updates the data structure. Unfortunately, *PlusCal* does not have nested processes, so we have to model sites and communicators as top-level processes. Sites are numbered from 1 to N, communicators from $N + 1$ to $2N$.

The constant *maxClock* is used to bound the state space explored by the model checker, see predicate *ClockConstraint* below.

$Sites \triangleq 1 \,..\, N$
$Comms \triangleq N + 1 \,..\, 2 * N$
$site(c) \triangleq c - N$ site a communicator is acting for
$max(x, y) \triangleq$ IF $x < y$ THEN y ELSE x

--algorithm *LamportMutex*

variables

two-dimensional array of message queues: enforce *FIFO* order between any pair of processes

$network = [from \in Sites \mapsto [to \in Sites \mapsto \langle\rangle]]$;

logical clock per site, initialized to 1

$clock = [s \in Sites \mapsto 1]$;

queue of pending requests per process, ordered by logical clock; entries are records of the form $[site \mapsto s,\ clk \mapsto c]$ where the clock value c is non-zero

$reqQ = [s \in Sites \mapsto \langle\rangle]$;

set of processes who sent acknowledgements for own request

$acks = [s \in Sites \mapsto \{\}]$;

define

check if request $rq1$ has higher priority than $rq2$ according to time stamp: both requests are records as they occur in *reqQ*

$beats(rq1, rq2) \triangleq$
$\lor\ rq1.clk < rq2.clk$
$\lor\ rq1.clk = rq2.clk \land rq1.site < rq2.site$

Compute the network obtained from net by sending message "*msg*" from site "from" to site "to". *NB*: Use a definition rather than a macro because this allows us to have multiple changes to the network in a single atomic step (rather kludgy, though).

$send(net, from, to, msg) \triangleq$
$[net$ EXCEPT $![from][to] = Append(@, msg)]$

Compute the network obtained from net by broadcasting message "*msg*" from site "from" to all sites.

$broadcast(net, from, msg) \triangleq$

```
    [net EXCEPT ![from] = [to ∈ Sites ↦ Append(net[from][to], msg)]]
end define ;

  insert a request from site from in reqQ of site s
macro insertRequest(s, from, clk)
begin
  with entry = [site ↦ from, clk ↦ clk],
       len = Len(reqQ[s]),
       pos = CHOOSE i ∈ 1 .. len + 1 :
                  ∧   ∀ j  ∈ 1 .. i − 1 : beats(reqQ[s][j], entry)
                  ∧    ∨ i = len + 1
                       ∨ beats(entry, reqQ[s][i])
   do
    reqQ[s] := SubSeq(reqQ[s], 1, pos − 1) ∘ ⟨entry⟩
                ∘       SubSeq(reqQ[s], pos, len) ;
  end with ;
end macro ;

remove a request from site from in reqQ of site s – assume that there is at most one such
request in the queue
macro removeRequest(s, from)
begin
  with len = Len(reqQ[s]),
       pos = CHOOSE i ∈ 1 .. len : reqQ[s][i].site = from
   do
    if (reqQ[s][pos].site = from)
     then
        request actually exists
       reqQ[s] := SubSeq(reqQ[s], 1, pos − 1) ∘ SubSeq(reqQ[s], pos + 1, len) ;
    end if   ;
  end with ;
end macro ;

process Site ∈ Sites
begin
  start:
    while TRUE
     do
  ncrit:
      skip ;
  try:
      network := broadcast(network, self,
                           [kind ↦ "request", clk ↦ clock[self]]) ;
      acks[self] := {} ;
  enter:
      await  ∧ Len(reqQ[self]) > 0
```

```
                    ∧ Head(reqQ[self]).site = self
                    ∧ acks[self] = Sites ;
      crit:
          skip ;
      exit:
          network := broadcast(network, self, [kind ↦ "free"]) ;
        end while ;
    end process ;

    process Comm ∈ Comms
    begin
      comm:
        while TRUE
         do
           pick some sender "from" and the oldest message sent from that node
          with me = site(self),
               from ∈ {s ∈ Sites : Len(network[s][me]) > 0},
               msg = Head(network[from][me]),
               _net = [network EXCEPT ![from][me] = Tail(@)]
           do
            if msg.kind = "request" then
              insertRequest(me, from, msg.clk) ;
              clock[me] := max(clock[me], msg.clk) + 1 ;
              network := send(_net, me, from, [kind ↦ "ack"]) ;
             elsif (msg.kind = "ack") then
              acks[me] := @ ∪ {from} ;
              network := _net ;
             elsif (msg.kind = "free") then
              removeRequest(me, from) ;
              network := _net ;
            end if   ;
          end with ;
        end while ;
    end process ;
  end algorithm
```

BEGIN TRANSLATION

VARIABLES $network$, $clock$, $reqQ$, $acks$, pc

define statement

$beats(rq1, rq2) \triangleq$
$\lor\ rq1.clk < rq2.clk$
$\lor\ rq1.clk = rq2.clk \land rq1.site < rq2.site$

$send(net,\ from,\ to,\ msg) \triangleq$
$[net \text{ EXCEPT } ![from][to] = Append(@,\ msg)]$

$broadcast(net,\ from,\ msg) \triangleq$
$[net \text{ EXCEPT } ![from] = [to \in Sites \mapsto Append(net[from][to],\ msg)]]$

$vars \triangleq \langle network,\ clock,\ reqQ,\ acks,\ pc \rangle$

$ProcSet \triangleq (Sites) \cup (Comms)$

$Init \triangleq$ Global variables
$\land\ network = [from \in Sites \mapsto [to \in Sites \mapsto \langle\rangle]]$
$\land\ clock = [s \in Sites \mapsto 1]$
$\land\ reqQ = [s \in Sites \mapsto \langle\rangle]$
$\land\ acks = [s \in Sites \mapsto \{\}]$
$\land\ pc = [self \in ProcSet \mapsto \text{CASE } self \in Sites \to \text{“start”}$
$\square\ self \in Comms \to \text{“comm”}]$

$start(self) \triangleq \land\ pc[self] = \text{“start”}$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“ncrit”}]$
$\land \text{ UNCHANGED } \langle network,\ clock,\ reqQ,\ acks \rangle$

$ncrit(self) \triangleq \land\ pc[self] = \text{“ncrit”}$
$\land \text{ TRUE}$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“try”}]$
$\land \text{ UNCHANGED } \langle network,\ clock,\ reqQ,\ acks \rangle$

$try(self) \triangleq \land\ pc[self] = \text{“try”}$
$\land\ network' = broadcast(network,\ self,$
$[kind \mapsto \text{“request”},\ clk \mapsto clock[self]])$
$\land\ acks' = [acks \text{ EXCEPT } ![self] = \{\}]$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“enter”}]$
$\land \text{ UNCHANGED } \langle clock,\ reqQ \rangle$

$enter(self) \triangleq \land\ pc[self] = \text{“enter”}$
$\land\ \land\ Len(reqQ[self]) > 0$
$\land\ Head(reqQ[self]).site = self$
$\land\ acks[self] = Sites$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“crit”}]$
$\land \text{ UNCHANGED } \langle network,\ clock,\ reqQ,\ acks \rangle$

$crit(self) \triangleq \land\ pc[self] = \text{“crit”}$
$\land \text{ TRUE}$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“exit”}]$
$\land \text{ UNCHANGED } \langle network,\ clock,\ reqQ,\ acks \rangle$

$$
\begin{array}{lcl}
exit(self) & \triangleq & \land\ pc[self] \ = \text{“exit”} \\
& & \land\ network' = broadcast(network,\ self,\ [kind \mapsto \text{“free”}]) \\
& & \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“start”}] \\
& & \land \text{ UNCHANGED } \langle clock,\ reqQ,\ acks \rangle \\
\\
Site(self) & \triangleq & start(self) \lor ncrit(self) \lor try(self) \lor enter(self) \\
& & \quad \lor crit(self) \lor exit(self) \\
\\
comm(self) & \triangleq & \land\ pc[self] = \text{“comm”} \\
& & \land \text{ LET } me \triangleq site(self) \text{ IN} \\
& & \quad \exists from \in \{s \in Sites : Len(network[s][me]) > 0\} : \\
& & \qquad \text{LET } msg \triangleq Head(network[from][me]) \text{ IN} \\
& & \qquad\quad \text{LET } \_net \triangleq [network \text{ EXCEPT } ![from][me] = Tail(@)] \text{ IN} \\
& & \qquad\qquad \text{IF } msg.kind = \text{“request”} \\
& & \qquad\qquad\quad \text{THEN } \land \text{ LET } entry \triangleq [site \mapsto from,\ clk \mapsto (msg.clk)] \text{ IN} \\
& & \qquad\qquad\qquad\quad \text{LET } len \triangleq Len(reqQ[me]) \text{ IN} \\
& & \qquad\qquad\qquad\qquad \text{LET } pos \triangleq \text{CHOOSE } i \in 1 \,..\, len + 1 : \\
& & \qquad\qquad\qquad\qquad\qquad \land \quad \forall j \ \in 1 \,..\, i - 1 : beats(reqQ[me][j],\ entry) \\
& & \qquad\qquad\qquad\qquad\qquad \land \quad \lor i = len + 1 \\
& & \qquad\qquad\qquad\qquad\qquad\qquad \lor beats(entry,\ reqQ[me][i]) \text{ IN} \\
& & \qquad\qquad\qquad\qquad\quad reqQ' = [reqQ \text{ EXCEPT } ![me] = SubSeq(reqQ[me],\ 1,\ pos - 1) \circ \langle e \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \circ \quad SubSeq(reqQ[me],\ pos,\ len \\
& & \qquad\qquad\qquad\quad \land\ clock' = [clock \text{ EXCEPT } ![me] = max(clock[me],\ msg.clk) + 1] \\
& & \qquad\qquad\qquad\quad \land\ network' = send(\_net,\ me,\ from,\ [kind \mapsto \text{“ack”}]) \\
& & \qquad\qquad\qquad\quad \land\ acks' = acks \\
& & \qquad\qquad\qquad \text{ELSE } \land \text{ IF } (msg.kind = \text{“ack”}) \\
& & \qquad\qquad\qquad\qquad \text{THEN } \land\ acks' = [acks \text{ EXCEPT } ![me] = @ \cup \{from\}] \\
& & \qquad\qquad\qquad\qquad\qquad \land\ network' = \_net \\
& & \qquad\qquad\qquad\qquad\qquad \land\ reqQ' = reqQ \\
& & \qquad\qquad\qquad\qquad \text{ELSE } \land \text{ IF } (msg.kind = \text{“free”}) \\
& & \qquad\qquad\qquad\qquad\qquad \text{THEN } \land \text{ LET } len \triangleq Len(reqQ[me]) \text{ IN} \\
& & \qquad\qquad\qquad\qquad\qquad\qquad \text{LET } pos \triangleq \text{CHOOSE } i \in 1 \,..\, len : reqQ[me][\\
& & \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{IF } (reqQ[me][pos].site = from) \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{THEN } \land\ reqQ' = [reqQ \text{ EXCEPT } ![me] \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE } \land \text{ TRUE} \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land\ reqQ' = reqQ \\
& & \qquad\qquad\qquad\qquad\qquad\quad \land\ network' = \_net \\
& & \qquad\qquad\qquad\qquad\qquad \text{ELSE } \land \text{ TRUE} \\
& & \qquad\qquad\qquad\qquad\qquad\qquad \land \text{ UNCHANGED } \langle network, \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad reqQ \rangle \\
& & \qquad\qquad\qquad\qquad\quad \land\ acks' = acks \\
& & \qquad\qquad\qquad\quad \land\ clock' = clock \\
& & \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“comm”}] \\
\\
Comm(self) & \triangleq & comm(self)
\end{array}
$$

$$Next \triangleq (\exists\, self \in Sites : Site(self)) \\ \quad \lor (\exists\, self \in Comms : Comm(self))$$

$$Spec \triangleq Init \land \Box[Next]_{vars}$$

END TRANSLATION

———————— definitions for verification ————————

constraint for bounding the state space during model checking

$$ClockConstraint \triangleq \forall\, s \in Sites : clock[s] \leq maxClock$$

set of possible messages exchanged between sites

$$Message \triangleq \\ [kind : \{\text{“request”}\},\ clk : Nat] \cup \{[kind \mapsto \text{“free”}],\ [kind \mapsto \text{“ack”}]\}$$

type invariant

$$TypeInvariant \triangleq \\ \land\ network \in [Sites \to [Sites \to Seq(Message)]] \\ \land\ clock \in [Sites \to Nat] \\ \land\ reqQ \in [Sites \to Seq([site : Sites,\ clk : Nat])] \\ \land \forall\, s \quad \in Sites : \forall\, i \in 1\,..\,Len(reqQ[s]) - 1 : beats(reqQ[s][i],\ reqQ[s][i+1]) \\ \land\ acks \in [Sites \to \text{SUBSET } Sites]$$

mutual exclusion

$$Mutex \triangleq \\ \forall\, s,\ t \in Sites : pc[s] = \text{“crit”} \land pc[t] = \text{“crit”} \Rightarrow s = t$$

other invariant properties: more complex to evaluate

$$Invariant \triangleq \\ \land \quad \text{each queue holds at most one request per site} \\ \quad \forall\, s \in Sites : \forall\, i \in 1\,..\,Len(reqQ[s]) : \\ \quad\ \forall\, j \in i+1\,..\,Len(reqQ[s]) : reqQ[s][j].site \neq reqQ[s][i].site \\ \land \quad \text{requests stay in queue until “free” message received} \\ \quad \forall\, s,\ t \in Sites : \\ \quad\ (\exists\, i \in 1\,..\,Len(network[s][t]) : network[s][t][i].kind = \text{“free”}) \\ \quad\ \Rightarrow \exists\, j \in 1\,..\,Len(reqQ[t]) : reqQ[t][j].site = s \\ \land \quad \text{site is in critical section only if at the head of every request queue} \\ \quad \forall\, s \in Sites : pc[s] = \text{“crit”} \Rightarrow \forall\, t \in Sites : Head(reqQ[t]).site = s$$

Fairness assumptions for proving liveness

$$Fairness \triangleq \\ \land \forall\, s \in Sites \quad : \mathrm{WF}_{vars}(enter(s)) \land \mathrm{WF}_{vars}(crit(s)) \land \mathrm{WF}_{vars}(exit(s)) \\ \land \forall\, c \in Comms : \mathrm{WF}_{vars}(comm(c))$$

$$FairSpec \triangleq Spec \land Fairness$$

$$Liveness \triangleq \forall\, s \in Sites : pc[s] = \text{“enter”} \rightsquigarrow pc[s] = \text{“crit”}$$

\ ∗ Modification History
\ ∗ Last modified Sun *Oct* 07 20:37:54 *EDT* 2018 by *mehuljain*
\ ∗ Created *Fri Oct* 05 20:38:41 *EDT* 2018 by *mehuljain*