MODULE *syncCon2*

EXTENDS *Integers*, *Sequences*, *FiniteSets*, *TLC*

CONSTANTS N, $FAILNUM$

ASSUME $N \leq 5 \land 0 \leq FAILNUM \land FAILNUM \leq 4$

$Nodes \triangleq 1 \,..\, N$

```
--algorithm syncCon1
{ variable FailNum = FAILNUM,
           up = [n ∈ Nodes ↦ TRUE] ;
           pt = [n ∈ Nodes ↦ 0] ;
           t = [n ∈ Nodes ↦ FALSE] ;
           d = [n ∈ Nodes ↦ -1] ;
           mb = [n ∈ Nodes ↦ {}] ;

  define {
    Choos the minimum value in a set
    SetMin(S) ≜ CHOOSE i ∈ S : ∀ j ∈ S : i ≤ j
    Identify the set of UP nodes
    UpNodes ≜ {n ∈ Nodes : up[n] = TRUE}
    Identify the number of up nodes
    ReturnUpNodes ≜ Cardinality(UpNodes)
  }
   This macro may fail a node or skip the value
  macro MayBeFail( ) {
    if ( FailNum > 0 ∧ up[self] ) {
         either {
                  up[self] := FALSE ;
                  FailNum := FailNum - 1 ;
                  }
         or skip ;
     } ;
  }

  fair process ( n ∈ Nodes )
   variables upNodes and tempNodes is used to identify the number of up nodes
   at the start and end of a round
  variable v = 0, pv = self, Q = {}, upNodes = ReturnUpNodes, tempNodes = upNodes ;
  {
  P: if ( up[self] ) {
       v := self ;
       Q := Nodes ;

          send vote to mb[p] one by one; this node can fail in between
  PS:    while ( up[self] ∧ Q ≠ {} ) {

           with ( p ∈ Q ) {
               Node can fail, such that up[self] will be set to False
```

```
        MayBeFail() ;
          pop p from the list of Nodes to broadcast to
        Q := Q \ {p} ;

          broadcast value only if my node is up
        if ( up[self] ) {
              assign value, this will be the min value received so far by node
            mb[p] := mb[p] ∪ {pv} ;
         } ;

      } ;   end-with

    } ;   end-while

    I may fail after broadcasting the value
    MayBeFail() ;

    if node is up then increment round
    if ( up[self] ) {
        pt[self] := pt[self] + 1 ;
    } ;

    to await, a process must be up and every other process should be on the same round
    if a process has rounds less than all other processes then it should advance
    say we have 4 processes and 1 fails, remaining all 3 process start at 0 and await till all are 1
    once all 3 are one, one of them advances and reaches at 2 but remaining processes wait to move to next
    round from 1 since all 3 process are not at same round, so choose if your round value is maximum or not
    if self value is minimum then stop awaiting else await
PR: await (up[self] = FALSE ∨ (up[self] ∧ (∀ i ∈ Nodes : IF up[i] THEN pt[i] ≥ pt[self] ELSE TRUE))) ;

    Number of up nodes at the end of one round
    tempNodes := ReturnUpNodes ;
    If the below conditions are true, node moves to the next round
    1. At the end of each round see if this is not the first round AND
    2. My node is up AND
    3. My node is not the only up node. AND
    4.(a) The number of upnodes has not changed during the course of the round OR
    4.(b) My number of rounds is strictly less than every other node which means
        that the number of rounds is not same for every node, node needs to move to the next round.
    if ( pt[self] ≠ 0 ∧ up[self] ∧ tempNodes > 1 ∧ ((upNodes > tempNodes) ∨ (∀ i ∈ Nodes : up[i] ⇒ pt[i
          record the minimum value for this round
        pv := SetMin(mb[self]) ;
          update the number of upNodes at the start of the next round
        upNodes := tempNodes ;
        goto P ;
    } else {
        If none of the conditions are true in the above if clause, go ahead and take
```

```
                decision
              goto D;
          } ;

          terminate and compute decision for up nodes
 D:     if ( up[self] ) {
          d[self] := SetMin(mb[self]);
          t[self] := TRUE;
        } ;
    } ;   end if
  }

   once a process takes a decision, it cannot be changed

}
```

BEGIN TRANSLATION

VARIABLES $FailNum$, up, pt, t, d, mb, pc

define statement

$SetMin(S) \triangleq$ CHOOSE $i \in S : \forall\, j \in S : i \leq j$

$UpNodes \triangleq \{n \in Nodes : up[n] = \text{TRUE}\}$

$ReturnUpNodes \triangleq Cardinality(UpNodes)$

VARIABLES v, pv, Q, $upNodes$, $tempNodes$

$vars \triangleq \langle FailNum,\ up,\ pt,\ t,\ d,\ mb,\ pc,\ v,\ pv,\ Q,\ upNodes,\ tempNodes\rangle$

$ProcSet \triangleq (Nodes)$

$Init \triangleq$ Global variables

$\land\ FailNum = FAILNUM$
$\land\ up = [n \in Nodes \mapsto \text{TRUE}]$
$\land\ pt = [n \in Nodes \mapsto 0]$
$\land\ t = [n \in Nodes \mapsto \text{FALSE}]$
$\land\ d = [n \in Nodes \mapsto -1]$
$\land\ mb = [n \in Nodes \mapsto \{\}]$

Process n

$\land\ v = [self \in Nodes \mapsto 0]$
$\land\ pv = [self \in Nodes \mapsto self]$
$\land\ Q = [self \in Nodes \mapsto \{\}]$
$\land\ upNodes = [self \in Nodes \mapsto ReturnUpNodes]$
$\land\ tempNodes = [self \in Nodes \mapsto upNodes[self]]$
$\land\ pc = [self \in ProcSet \mapsto \text{“P”}]$

$P(self) \triangleq \land\ pc[self] = \text{“P”}$
$\qquad\qquad \land\ \text{IF}\ up[self]$

```
                THEN ∧ v' = [v EXCEPT ![self] = self]
                     ∧ Q' = [Q EXCEPT ![self] = Nodes]
                     ∧ pc' = [pc EXCEPT ![self] = "PS"]
                ELSE ∧ pc' = [pc EXCEPT ![self] = "Done"]
                     ∧ UNCHANGED ⟨v, Q⟩
          ∧ UNCHANGED ⟨FailNum, up, pt, t, d, mb, pv, upNodes, tempNodes⟩

PS(self) ≜ ∧ pc[self] = "PS"
           ∧ IF up[self] ∧ Q[self] ≠ {}
                THEN ∧ ∃ p ∈ Q[self] :
                          ∧ IF FailNum > 0 ∧ up[self]
                               THEN ∧ ∨ ∧ up' = [up EXCEPT ![self] = FALSE]
                                        ∧ FailNum' = FailNum − 1
                                     ∨ ∧ TRUE
                                        ∧ UNCHANGED ⟨FailNum, up⟩
                               ELSE ∧ TRUE
                                    ∧ UNCHANGED ⟨FailNum, up⟩
                          ∧ Q' = [Q EXCEPT ![self] = Q[self] \ {p}]
                          ∧ IF up'[self]
                               THEN ∧ mb' = [mb EXCEPT ![p] = mb[p] ∪ {pv[self]}]
                               ELSE ∧ TRUE
                                    ∧ mb' = mb
                     ∧ pc' = [pc EXCEPT ![self] = "PS"]
                     ∧ pt' = pt
                ELSE ∧ IF FailNum > 0 ∧ up[self]
                          THEN ∧ ∨ ∧ up' = [up EXCEPT ![self] = FALSE]
                                   ∧ FailNum' = FailNum − 1
                                ∨ ∧ TRUE
                                   ∧ UNCHANGED ⟨FailNum, up⟩
                          ELSE ∧ TRUE
                               ∧ UNCHANGED ⟨FailNum, up⟩
                     ∧ IF up'[self]
                          THEN ∧ pt' = [pt EXCEPT ![self] = pt[self] + 1]
                          ELSE ∧ TRUE
                               ∧ pt' = pt
                     ∧ pc' = [pc EXCEPT ![self] = "PR"]
                     ∧ UNCHANGED ⟨mb, Q⟩
           ∧ UNCHANGED ⟨t, d, v, pv, upNodes, tempNodes⟩

PR(self) ≜ ∧ pc[self] = "PR"
           ∧ (up[self] = FALSE ∨ (up[self] ∧ (∀ i ∈ Nodes : IF up[i] THEN pt[i] ≥ pt[self] ELSE TRUE)))
           ∧ tempNodes' = [tempNodes EXCEPT ![self] = ReturnUpNodes]
           ∧ IF pt[self] ≠ 0 ∧ up[self] ∧ tempNodes'[self] > 1 ∧ ((upNodes[self] > tempNodes'[self]) ∨ (∀ i ∈
                THEN ∧ pv' = [pv EXCEPT ![self] = SetMin(mb[self])]
                     ∧ upNodes' = [upNodes EXCEPT ![self] = tempNodes'[self]]
```

$$
\begin{array}{rl}
& \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“P”}] \\
\text{ELSE} & \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“D”}] \\
& \land \text{ UNCHANGED } \langle pv,\ upNodes \rangle \\
\land \text{ UNCHANGED } & \langle FailNum,\ up,\ pt,\ t,\ d,\ mb,\ v,\ Q \rangle
\end{array}
$$

$$
\begin{array}{rl}
D(self) \triangleq & \land\ pc[self] = \text{“D”} \\
& \land \text{ IF } up[self] \\
& \quad \text{THEN } \land\ d' = [d \text{ EXCEPT } ![self] = SetMin(mb[self])] \\
& \qquad\ \ \land\ t' = [t \text{ EXCEPT } ![self] = \text{TRUE}] \\
& \quad \text{ELSE } \land \text{ TRUE} \\
& \qquad\ \ \land \text{ UNCHANGED } \langle t,\ d \rangle \\
& \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Done”}] \\
& \land \text{ UNCHANGED } \langle FailNum,\ up,\ pt,\ mb,\ v,\ pv,\ Q,\ upNodes,\ tempNodes \rangle
\end{array}
$$

$$n(self) \triangleq P(self) \lor PS(self) \lor PR(self) \lor D(self)$$

$$
\begin{array}{rl}
Next \triangleq & (\exists\, self \in Nodes : n(self)) \\
& \lor \quad \text{Disjunct to prevent deadlock on termination} \\
& \quad ((\forall\, self \in ProcSet : pc[self] = \text{“Done”}) \land \text{UNCHANGED } vars)
\end{array}
$$

$$
\begin{array}{rl}
Spec \triangleq & \land\ Init \land \Box[Next]_{vars} \\
& \land\ \forall\, self \in Nodes : \mathrm{WF}_{vars}(n(self))
\end{array}
$$

Agreement invariant
The nodes that have terminated (UP nodes) have the same decision values

$$Agreement \triangleq \forall\, i \in Nodes : \forall\, j \in Nodes : \text{IF } t[i] \land t[j] \text{ THEN } d[i] = d[j] \text{ ELSE TRUE}$$

Validation Property

$$Validity \triangleq (\exists\, k \in Nodes : \forall\, i \in Nodes : i = k) \Rightarrow (\forall\, i \in Nodes : \text{IF } t[i] \text{ THEN } d[i] = i \text{ ELSE TRUE})$$

Termination property
If the node is UP, it has terminated (its t is TRUE) otherwise it has not teminated

$$Termination \triangleq \Diamond(\forall\, i \in Nodes : \text{IF } up[i] \text{ THEN } t[i] = \text{TRUE ELSE } t[i] = \text{FALSE})$$

END TRANSLATION

END TRANSLATION

Members:

Name: *Sneha Mehta UBIT* Name: snehameh Person $\neq -50245877$

Name: *Varun Jain UBIT* Name: varunjai Person $\neq -50247176$

Explaination

This program acheives consensus with crash faults. We tested the above algorithm with:
1. 3 nodes and 1 crash fault

2. 4 nodes and 1 crash fault
3. 4 nodes and 2 crash faults

In this program,

– Each node maintains the number of UP nodes at the beginning of a round.
- Each node waits for completion of a round.
- Each node checks the difference in the number of UP nodes at the completion of a round.
- Each node also checks if it is at the same number of rounds as the other nodes.
- In case a node encounters that the number of up nodes at the beginning of a round differs from the number of nodes post completion of the round then it knows that some node crashed and it should move to another round to achieve consensus.

We check the program agains the below properties Agreement Property: The nodes that have terminated (UP nodes) have the same decision values.

Termination Property: If the node is UP, it has terminated (its t is TRUE) otherwise it has not teminated.

Validation Property If all initial values are equal, correct processes must decide on that value.