MODULE *syncCon2*

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

CONSTANT N, *FAILNUM*

ASSUME $N \leq 5 \ \land 0 \leq FAILNUM \land FAILNUM \leq 2$

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

```
--algorithm syncCon2
{
    variable FailNum = FAILNUM,
             up = [n ∈ Nodes ↦ TRUE],
             pt = [n ∈ Nodes ↦ 0],
             t = [n ∈ Nodes ↦ FALSE],
             d = [n ∈ Nodes ↦ -1],
             mb = [n ∈ Nodes ↦ {}],
             Fail = FailNum ;

    define {
    SetMin(S) ≜ CHOOSE i ∈ S : ∀ j ∈ S : i ≤ j
    UpNodes ≜ {n ∈ Nodes : up[n] = TRUE}
    CheckNodesRound(n) ≜  {∀ i ∈ UpNodes : pt[i] = pt[n]}

    }

    macro MaybeFail( ) {
        if ( FailNum > 0 ∧ up[self] )
            { either
                { up[self] := FALSE ; FailNum := FailNum - 1 ; }
                or skip ; } ;

    }

    fair process ( n ∈ Nodes )
    variable v = 0, pv = 0, Q = {}, pmb_count = 0, r = 1 ;      \* "pmb_count" is used to keep the info of
    {                                                           \* "pv" holds the previous minimum value
                                                                \* "r" is the round counter needed for the
P:  while ( up[self] ∧ r > 0 ) {          \* next round is executed when node is up and round counter is set to 0

        if ( pt[self] = 0 ) { v := self ; } ;      \* set the v to self for first round else the previous decision
        else { v := d[self] } ;
        Q := Nodes ;                               \* set the Q to all nodes for sending the message
PS: while ( up[self] ∧ Q ≠ {} ) {
        with ( p ∈ Q ) {
              mb[p] := mb[p] ∪ {v} ;   \* Append the value of self nede to message bus of p node
              Q := Q \ {p} ;                \* Remove the p from Q to send message to other nodes
              MaybeFail() ;            \* Call fail macro to fail the node if FailNum has not reached the maximum limit
         } ;
    } ;
```

```
    if ( up[self] ) pt[self] := pt[self] + 1 ;   \* Move to next round

PR: await (up[self] ∧ (∀ k ∈ UpNodes : pt[self] ≤ pt[k])) ;          \* Execute receive step only if the node
    {
        pv := d[self] ;                          \* stor the value of previous decision in pv
        d[self] := SetMin(mb[self]) ;    \* Set the decision to the minimum value received by all the nodes

          \* the below condition checks for message count in previous round and current round. Also, the if the decision varies
          \* if any of the condition differs I need another round to make sure I get the minimum value
        if ( pmb_count ≠ Cardinality(mb[self]) ∨ pv ≠ d[self] ) r := 1 ;
        else r := r − 1 ;

        pmb_count := Cardinality(mb[self]) ;    \* set the message count of current step to pmb_count
        mb[self] := {} ;                        \* clear the mailbox for next round

        if ( r = 0 ) t[self] := TRUE ;          \* when no more rounds are needed for the current node terminate the

    } await

    } while

    } process

} \* algorithm
```

BEGIN TRANSLATION

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

define statement

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

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

$CheckNodesRound(n) \triangleq \{\forall i \in UpNodes : pt[i] = pt[n]\}$

VARIABLES v, pv, Q, pmb_count, r

$vars \triangleq \langle FailNum, up, pt, t, d, mb, Fail, pc, v, pv, Q, pmb\_count, r\rangle$

$ProcSet \triangleq (Nodes)$

$Init \triangleq$ Global variables

$\land\ FailNum = FAILNUM$

$\land\ up = [n \in Nodes \mapsto \text{TRUE}]$

$\land\ pt = [n \in Nodes \mapsto 0]$

$\land\ t = [n \in Nodes \mapsto \text{FALSE}]$

$\land\ d = [n \in Nodes \mapsto -1]$

$\land\ mb = [n \in Nodes \mapsto \{\}]$

$\land\ Fail = FailNum$

Process n

$\land\ v = [self \in Nodes \mapsto 0]$

$\land\ pv = [self \in Nodes \mapsto 0]$
$\land\ Q = [self \in Nodes \mapsto \{\}]$
$\land\ pmb\_count = [self \in Nodes \mapsto 0]$
$\land\ r = [self \in Nodes \mapsto 1]$
$\land\ pc = [self \in ProcSet \mapsto \text{“P”}]$

$P(self) \triangleq$
$\land\ pc[self] = \text{“P”}$
$\land$ IF $up[self] \land r[self] > 0$
THEN $\land$ IF $pt[self] = 0$
THEN $\land\ v' = [v$ EXCEPT $![self] = self]$
ELSE $\land\ v' = [v$ EXCEPT $![self] = d[self]]$
$\land\ Q' = [Q$ EXCEPT $![self] = Nodes]$
$\land\ pc' = [pc$ EXCEPT $![self] = \text{“PS”}]$
ELSE $\land\ pc' = [pc$ EXCEPT $![self] = \text{“Done”}]$
$\land$ UNCHANGED $\langle v, Q\rangle$
$\land$ UNCHANGED $\langle FailNum, up, pt, t, d, mb, Fail, pv, pmb\_count, r\rangle$

$PS(self) \triangleq$
$\land\ pc[self] = \text{“PS”}$
$\land$ IF $up[self] \land Q[self] \neq \{\}$
THEN $\land\ \exists\, p \in Q[self]:$
$\land\ mb' = [mb$ EXCEPT $![p] = mb[p] \cup \{v[self]\}]$
$\land\ Q' = [Q$ EXCEPT $![self] = Q[self] \setminus \{p\}]$
$\land$ IF $FailNum > 0 \land up[self]$
THEN $\land\ \lor\ \land\ up' = [up$ EXCEPT $![self] =$ FALSE$]$
$\land\ FailNum' = FailNum - 1$
$\lor\ \land$ TRUE
$\land$ UNCHANGED $\langle FailNum, up\rangle$
ELSE $\land$ TRUE
$\land$ UNCHANGED $\langle FailNum, up\rangle$
$\land\ pc' = [pc$ EXCEPT $![self] = \text{“PS”}]$
$\land\ pt' = pt$
ELSE $\land$ IF $up[self]$
THEN $\land\ pt' = [pt$ EXCEPT $![self] = pt[self] + 1]$
ELSE $\land$ TRUE
$\land\ pt' = pt$
$\land\ pc' = [pc$ EXCEPT $![self] = \text{“PR”}]$
$\land$ UNCHANGED $\langle FailNum, up, mb, Q\rangle$
$\land$ UNCHANGED $\langle t, d, Fail, v, pv, pmb\_count, r\rangle$

$PR(self) \triangleq$
$\land\ pc[self] = \text{“PR”}$
$\land\ (up[self] \land (\forall\, k \in UpNodes : pt[self] \leq pt[k]))$
$\land\ pv' = [pv$ EXCEPT $![self] = d[self]]$
$\land\ d' = [d$ EXCEPT $![self] = SetMin(mb[self])]$
$\land$ IF $pmb\_count[self] \neq Cardinality(mb[self]) \lor pv'[self] \neq d'[self]$
THEN $\land\ r' = [r$ EXCEPT $![self] = 1]$
ELSE $\land\ r' = [r$ EXCEPT $![self] = r[self] - 1]$

$\land\ pmb\_count' = [pmb\_count \text{ EXCEPT } ![self] = Cardinality(mb[self])]$
$\land\ mb' = [mb \text{ EXCEPT } ![self] = \{\}]$
$\land$ IF $r'[self] = 0$
THEN $\land\ t' = [t \text{ EXCEPT } ![self] = \text{TRUE}]$
ELSE $\land$ TRUE
$\land\ t' = t$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“P”}]$
$\land$ UNCHANGED $\langle FailNum,\ up,\ pt,\ Fail,\ v,\ Q\rangle$

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

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

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

$Termination \triangleq \Diamond(\forall\, self \in ProcSet : pc[self] = \text{“Done”})$

END TRANSLATION
* Below property is for termination
$FinalState \triangleq \text{TRUE} \rightsquigarrow (\forall\, i \in Nodes : up[i] = \text{TRUE} \Rightarrow t[i] = \text{TRUE})$

---

* Below are the invariants
$Agg \triangleq \forall\, i, j \in Nodes : (t[i] \land t[j]) \Rightarrow (d[i] = d[j])$
$Validity \triangleq (\exists\, k \in Nodes : (\forall\, i \in Nodes : v[i] = k)) \Rightarrow (\forall\, j \in Nodes : t[j] = \text{TRUE} \Rightarrow d[j] = v[j])$

$Inv \triangleq Agg \land Validity$

---

* Modification History
* Last modified *Tue Oct* 24 23:43:54 *EDT* 2017 by Deep
* Created *Tue Oct* 24 20:54:28 *EDT* 2017 by Deep

* DEEP *NARAYAN MISHRA − PERSON* NO 50245878

* As we know the *Aggrement* property and validitiy will fail if there is any node failure while sending
* the message to other nodes. Single round will not help to reach the consensue by all the parties.
*
*
*
* In brief - we need to run for multiple rounds to establish the consences. However, every nodes need
* to identify when to go for the next round and when to stop.
*

* This problem can be solved by observing the behaviour of nodes' mail box and the decision in any two
* consecutive rounds of the node. If we closly monitor both these properties. If there is any failure the message count received
* by a node will be lesser than the previous round. Or the decision made in previous round would be different from
* current round decision of a node.
* In general, either of the parameters will differ is there is any failure. By looking at the difference in both the
* property (mail box size, and decisions) of the node we can identify whether to go for next round or not.
*
*
*
* Based on the above analogy, I have two extra varibles (pmb_count : previous mail box count) and (pv : previous
* decisions). If there is difference in any of these parameters I am setting the round marker (r: round count) to 1.
* If none of the prameters changes we are assure of no failure and no further rounds need. Hence setting r to 0.