MODULE *ConsensusPlusCal*

```
EXTENDS $Integers, Sequences, TLC$
CONSTANTS
    $Names,$
    $PossibleAllocations,$
    $Participants,$
    $Allocations,$
    $NULL$
ASSUME $Len(Participants) = Len(Allocations)$

$NumParticipants \triangleq Len(Participants)$

  --algorithm consensus_update

 For the moment, we assume that participants only send commitments forward.
 Since messages are read and then discarded, it's enough to just store one.
variables $msg = NULL$

define
$ourTurn \triangleq$ TRUE
$allocationOk \triangleq$ TRUE
end define ;

fair process $updateConsensus \in$ DOMAIN $Participants$
variable
  $state = [allocation \mapsto Allocations[self], turnNumber \mapsto 1, type \mapsto$ "Waiting"$]$,
  $me = Participants[self]$
begin
    Each participant atomically reads the message, updates their state,
    and sends a message if it's their turn, accordingly.
    We assume that messages that create invalid transitions are discarded.
    Therefore, every incoming message considered here is considered a source of truth.
  $A$:
    if
       $\land\ msg \neq NULL$
       $\land\ msg.to = me$
       $\land\ msg.turnNumber > state.turnNumber$
     then
          First, update our state based on the incoming message
        if $msg.furtherVotesRequired = 0$
         then $state := [type \mapsto$ "Success"$]$ ;
         elsif $ourTurn$
         then
             if $state.type =$ "Sent"
              then $state := [type \mapsto$ "Failure"$]$ ;
```

```
            elsif allocationOk
            then skip ;   TODO: Send vote
            else   skip ;   TODO: Send reject
            end if ;
        else
            state := [
                allocation    ↦ state.allocation,
                turnNumber ↦ msg.turnNumber,
                type             ↦ "Waiting"
            ] ;
        end if ;
    end if ;
end process ;

end algorithm   ;
 BEGIN TRANSLATION
VARIABLES msg, pc

 define statement
ourTurn ≜ TRUE
allocationOk ≜ TRUE

VARIABLES state, me

vars ≜ ⟨msg, pc, state, me⟩

ProcSet ≜ (DOMAIN Participants)

Init ≜   Global variables
        ∧ msg = NULL
         Process updateConsensus
        ∧ state = [self ∈ DOMAIN Participants ↦ [allocation ↦ Allocations[self], turnNumber ↦ 1, type ↦ "
        ∧ me = [self ∈ DOMAIN Participants ↦ Participants[self]]
        ∧ pc  = [self ∈ ProcSet ↦ "A"]

A(self) ≜  ∧ pc[self] = "A"
           ∧ IF  ∧ msg ≠ NULL
                 ∧ msg.to = me[self]
                 ∧ msg.turnNumber > state[self].turnNumber
                 THEN  ∧ IF msg.furtherVotesRequired = 0
                            THEN  ∧ state′ = [state EXCEPT ![self] = [type ↦ "Success"]]
                            ELSE  ∧ IF ourTurn
                                       THEN  ∧ IF state[self].type = "Sent"
                                                  THEN  ∧ state′ = [state EXCEPT ![self] = [type ↦ "Fail
                                                  ELSE  ∧ IF allocationOk
                                                             THEN  ∧ TRUE
                                                             ELSE  ∧ TRUE
```

```
                                    $\land\ state' = state$
                  ELSE  $\land\ state' = [state$ EXCEPT $![self] =$      [
                                                                   $allocation \mapsto state[se$
                                                                   $turnNumber \mapsto msg.tu$
                                                                   $type \mapsto$ “Waitin
                                                               ]]
        ELSE  $\land$ TRUE
              $\land\ state' = state$
    $\land\ pc' = [pc$ EXCEPT $![self] =$ “Done”$]$
    $\land$ UNCHANGED $\langle msg, me\rangle$
```

$updateConsensus(self) \triangleq A(self)$

Allow infinite stuttering to prevent deadlock on termination.

```
$Terminating \triangleq \land\ \forall\, self \in ProcSet : pc[self] =$ “Done”
                 $\land$ UNCHANGED $vars$

$Next \triangleq (\exists\, self \in$ DOMAIN $Participants : updateConsensus(self))$
           $\lor\ Terminating$

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

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

END TRANSLATION

```
$AllowedMessages \triangleq$
  [
    $turnNumber : Nat,$
    $votesRequired : 0 \,..\, (NumParticipants - 1),$
    $to : Names,$
    $allocation : PossibleAllocations$
  ]
  $\cup \{NULL\}$

$States \triangleq \{\}$
  $\cup\ [allocation : PossibleAllocations,\ turnNumber : Nat,\ type : \{$“Waiting”$\}]$
  $\cup\ [allocation : PossibleAllocations,\ turnNumber : Nat,\ type : \{$“Sent”$\},\ status : \{$“Voted”, “Rejected”$\}]$

$TypeOK \triangleq$
  $\land\ PrintT(\langle msg, state\rangle)$  Debugging statement
  The following two conditions specify the format of each message and
  participant state
  $\land\ state \in [$DOMAIN $Participants \rightarrow States]$
  $\land\ msg \in AllowedMessages$
```

```
$TurnNumberIncrements \triangleq$
  $\land \forall p \in \textsc{domain}\ Participants : state'[p].turnNumber \geq state[p].turnNumber$

$ProtocolTerminates \triangleq \Diamond\Box($
  $\forall p \in \textsc{domain}\ Participants :$
      $\lor state[p].type = \text{“Success”}$
      $\lor state[p].type = \text{“Failure”}$
$)$
```

* Modification History
* Last modified *Wed Aug* 07 09:27:25 *MDT* 2019 by *andrewstewart*
* Created *Tue Aug* 06 14:38:11 *MDT* 2019 by *andrewstewart*