MODULE *ConsensusUpdate*

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

CONSTANTS

Names, a set

Participants, an array of participants, in their order in the state channel

NULL

ASSUME

$\land\ Len(Participants) > 1$

$NumParticipants \triangleq Len(Participants)$

$Types \triangleq [$
$\quad WAITING \mapsto$ "WAITING",
$\quad SENT \mapsto$ "SENT",
$\quad SUCCESS \mapsto$ "SUCCESS",
$\quad FAILURE \mapsto$ "FAILURE"
$]$

$Status \triangleq [$
$\quad OK \mapsto$ "OK",
$\quad ABORT \mapsto$ "ABORT",
$\quad SUCCESS \mapsto$ "SUCCESS"
$]$

$Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$
$Running(state) \triangleq state.type \in \{Types.WAITING, Types.SENT\}$
$Terminated(state) \triangleq \neg Running(state)$

--algorithm *consensus_update*

For the moment, we assume that participants only send commitments forward.
Since a read message is then discarded, it's enough to just store one.

variables $msgs = [p \in Names \mapsto \{\}]$

define

Arrays are 1-indexed, while the % operator returns a number between 0 and *NumParticipants*.
This explains the following slightly complicated expression

$mover(turnNumber) \triangleq 1 + ((turnNumber - 1)\%NumParticipants)$
$currentlyOurTurn(state) \triangleq$
$\quad \land\ state.type = Types.WAITING$
$\quad \land\ state.ourIndex = mover(state.turnNumber)$
$becomesOurTurn(state, msg) \triangleq$
$\quad \land\ state.type = Types.WAITING$
$\quad \land\ msg.status = Status.OK$
$\quad \land\ state.ourIndex = mover(msg.turnNumber)$

$target(turnNumber) \triangleq Participants[mover(turnNumber)]$

end define ;

```
macro pushMsg(m, sender)
begin
if msg = NULL then
    msgs := [recipient ∈ Names ↦ IF recipient = sender THEN msgs[recipient] ELSE  msgs[recipient] ∪ {m}] ;
 else
    msgs := [recipient ∈ Names ↦ IF recipient = sender THEN msgs[recipient] \ {msg} ELSE  msgs[recipient] U
end if ;
msg := NULL
end macro ;

macro sendVote(turnNumber, votesRequired, me)
begin
assert votesRequired > 0 ;
state := [
    type ↦ Types.SENT,
    turnNumber ↦ turnNumber,
    ourIndex     ↦ state.ourIndex
] ;
pushMsg([
    turnNumber       ↦ state.turnNumber,
    votesRequired    ↦ votesRequired,
    status           ↦ Status.OK
], me)
end macro ;


macro returnSuccess(me)
begin
state := [type     ↦ Types.SUCCESS] @@ state ;
pushMsg([status ↦ Status.SUCCESS], me)
end macro ;

macro returnFailure(turnNumber, me)
begin
state := [
    type ↦ Types.FAILURE,
    turnNumber ↦ turnNumber
] @@ state ;
pushMsg([
    status ↦ Status.ABORT
], me) ;
end macro ;

macro vote(turnNumber, votesRequired)
begin
if votesRequired = 0 then returnSuccess(me)
```

```
 else sendVote(turnNumber, votesRequired, me)
end if ; end macro ;

macro waitForUpdate(turnNumber, me)
begin
state := [
    turnNumber ↦ msg.turnNumber,
    type       ↦ Types.WAITING,
    ourIndex   ↦ state.ourIndex
] ;
msgs[me] := msgs[me] \ {msg} ;
end macro ;
```

Calling this a fair process prevents the process from stuttering forever. It's always considered to be valid to take a step where your state variables don't change, which could be the case if some unrelated protocols end up in an infinite loop, for instance. However, we want to check that IF A: wallets always eventually take some valid action THEN B: wallets always eventually terminate the consensus-update protocol Calling the process fair ensures that A is true, and therefore the model checks that under the assumption A, B is also true.

```
fair process consensusUpdate ∈ DOMAIN Participants
variables
  state = [
    turnNumber ↦ 1,
    ourIndex ↦ self,
    type ↦ Types.WAITING
  ],
  me = Participants[self],
  msg = NULL

begin
```

Each participant either sends a message if it's currently safe to do so, or else it reads a message for the participant, updates their state accordingly, and sends a message if it's then safe. These actions are currently assumed to be atomic, and are therefore assigned to a single label, *ReachConsensus*

```
  ReachConsensus:
    while Running(state) do
        if currentlyOurTurn(state) then
            if state.type = Types.WAITING then vote(state.turnNumber + 1, NumParticipants − 1) ;
             elsif state.type = Types.SENT  then returnFailure(state.turnNumber, me) ;
             else assert FALSE
            end if ;
         else
        ReadMessage:
            with m ∈ msgs[me] do msg := mend with ;
        ProcessMessage:
            If the commitment receved is not valid, return FAILURE
            TODO : Is this the actual behaviour we want?
```

```
            In the readme, we say this is what works, but the reducer does not
            work this way
          either returnFailure(state.turnNumber, me)
            In this case, the commitment was valid.
          or if msg.status = Status.OK then
              if msg.turnNumber > state.turnNumber then
                  First, update our state based on the incoming message
                 if msg.votesRequired = 0 then returnSuccess(me)
                  elsif becomesOurTurn(state, msg) then   TODO : This should check if we re moving after receiv
                     if    state.type  = Types.SENT then returnFailure(msg.turnNumber, me)
                      elsif state.type = Types.WAITING then vote(msg.turnNumber + 1, msg.votesRequire
                      else assert FALSE ;
                     end if ;
                  else
                     waitForUpdate(msg, me) ;
                 end if ;
              end if ;
           elsif msg.status = Status.ABORT    then returnFailure(state.turnNumber, me)
           elsif msg.status = Status.SUCCESS then returnSuccess(me)
          end if ; end either ;
        end if ;

    end while ;
end process ;

end algorithm  ;
 BEGIN TRANSLATION
VARIABLES msgs, pc

 define statement
mover(turnNumber) ≜ 1 + ((turnNumber − 1)%NumParticipants)
currentlyOurTurn(state) ≜
    ∧ state.type = Types.WAITING
    ∧ state.ourIndex = mover(state.turnNumber)
becomesOurTurn(state, msg) ≜
   ∧ state.type  = Types.WAITING
   ∧ msg.status = Status.OK
   ∧ state.ourIndex = mover(msg.turnNumber)




target(turnNumber) ≜ Participants[mover(turnNumber)]

VARIABLES state, me, msg
```

$vars \triangleq \langle msgs,\ pc,\ state,\ me,\ msg \rangle$

$ProcSet \triangleq (\text{DOMAIN } Participants)$

$Init \triangleq$ Global variables
$\land\ msgs = [p \in Names \mapsto \{\}]$
Process *consensusUpdate*
$\land\ state = [self \in \text{DOMAIN } Participants \mapsto$ [
$turnNumber \mapsto 1,$
$ourIndex \mapsto self,$
$type \mapsto Types.WAITING$
]]
$\land\ me = [self \in \text{DOMAIN } Participants \mapsto Participants[self]]$
$\land\ msg = [self \in \text{DOMAIN } Participants \mapsto NULL]$
$\land\ pc = [self \in ProcSet \mapsto$ "ReachConsensus"$]$

$ReachConsensus(self) \triangleq$ $\land\ pc[self] =$ "ReachConsensus"
$\land$ IF $Running(state[self])$
THEN $\land$ IF $currentlyOurTurn(state[self])$
THEN $\land$ IF $state[self].type = Types.WAITING$
THEN $\land$ IF $(NumParticipants - 1) = 0$
THEN $\land\ state' = [state$ EXCEPT $![self] =$
$\land$ IF $msg[self] = NULL$
THEN $\land\ msgs' = [recipient$
ELSE $\land\ msgs' = [recipient$
$\land\ msg' = [msg$ EXCEPT $![self] =$
ELSE $\land\ Assert((NumParticipants - 1)$
"Failure of assertion at li
$\land\ state' = [state$ EXCEPT $![self] =$
$\land$ IF $msg[self] = NULL$
THEN $\land\ msgs' = [recipient$
$turnN$
$votesR$
$status$
])}]
ELSE $\land\ msgs' = [recipient$
$turnN$
$votesR$
$status$
])}]
$\land\ msg' = [msg$ EXCEPT $![self] =$
ELSE $\land$ IF $state[self].type = Types.SENT$

```
                                                                THEN  ∧ state′ = [state EXCEPT ![self] =

                                                                                                  ∧ IF msg[self] = NULL
                                                                                                        THEN  ∧ msgs′ = [recipient
                                                                                                                         status
                                                                                                                        ])}]
                                                                                                        ELSE  ∧ msgs′ = [recipient
                                                                                                                         status
                                                                                                                        ])}]
                                                                                                  ∧ msg′ = [msg EXCEPT ![self] =
                                                                                        ELSE  ∧ Assert(FALSE,
                                                                                                       "Failure of assertion at li
                                                                                                  ∧ UNCHANGED ⟨msgs,
                                                                                                               state,
                                                                                                               msg⟩
                                                                                 ∧ pc′ = [pc EXCEPT ![self] = "ReachConsensus"]
                                                                           ELSE  ∧ pc′ = [pc EXCEPT ![self] = "ReadMessage"]
                                                                                 ∧ UNCHANGED ⟨msgs, state, msg⟩
                                                                     ELSE  ∧ pc′ = [pc EXCEPT ![self] = "Done"]
                                                                           ∧ UNCHANGED ⟨msgs, state, msg⟩
                                                               ∧ me′ = me

ReadMessage(self) ≜  ∧ pc[self] = "ReadMessage"
                     ∧ ∃ m ∈ msgs[me[self]] :
                         msg′ = [msg EXCEPT ![self] = m]
                     ∧ pc′ = [pc EXCEPT ![self] = "ProcessMessage"]
                     ∧ UNCHANGED ⟨msgs, state, me⟩

ProcessMessage(self) ≜  ∧ pc[self] = "ProcessMessage"
                        ∧ ∨ ∧ state′ = [state EXCEPT ![self] =          [
                                                                  type ↦ Types.FAILURE,
                                                                  turnNumber ↦ (state[self].turnNumber)
                                                                ] @@ state[self]]
                                                          ∧ IF msg[self] = NULL
                                                                THEN  ∧ msgs′ = [recipient ∈ Names ↦ IF recipient = me[self] THEN ms
                                                                                    status ↦ Status.ABORT
                                                                               ])}]
                                                                ELSE  ∧ msgs′ = [recipient ∈ Names ↦ IF recipient = me[self] THEN ms
                                                                                    status ↦ Status.ABORT
                                                                               ])}]
                                                          ∧ msg′ = [msg EXCEPT ![self] = NULL]
                          ∨ ∧ IF msg[self].status = Status.OK
                                THEN  ∧ IF msg[self].turnNumber > state[self].turnNumber
```

```
THEN ∧ IF msg[self].votesRequired = 0
          THEN ∧ state′ = [state EXCEPT ![self] = [type ↦
               ∧ IF msg[self] = NULL
                    THEN ∧ msgs′ = [recipient ∈ Name
                    ELSE ∧ msgs′ = [recipient ∈ Name
               ∧ msg′ = [msg EXCEPT ![self] = NULL]
          ELSE ∧ IF becomesOurTurn(state[self], msg[sel
                    THEN ∧ IF state[self].type = Types.
                              THEN ∧ state′ = [state
                         ∧ IF msg[self] =
                              THEN ∧ m
                              ELSE ∧ m
                         ∧ msg′ = [msg E
               ELSE ∧ IF state[self].t
                    THEN ∧ IF
```

```
                                                            ELSE  ∧ A
                                                                  ∧ UN
                                              ELSE  ∧ state' = [state EXCEPT ![se
                                                    ∧ msgs' = [msgs EXCEPT ![m
                                                    ∧ msg' = msg
                                  ELSE  ∧ TRUE
                                        ∧ UNCHANGED ⟨msgs,
                                                     state, msg⟩
                             ELSE  ∧ IF msg[self].status = Status.ABORT
                                  THEN  ∧ state' = [state EXCEPT ![self] =          [
                                                                      type ↦ Types.FA
                                                                      turnNumber ↦ (
                                                                    ] @@ state[self]]
                                        ∧ IF msg[self] = NULL
                                             THEN  ∧ msgs' = [recipient ∈ Names ↦ IF recip
                                                                   status ↦ Status.ABORT
                                                               ])}]
                                             ELSE  ∧ msgs' = [recipient ∈ Names ↦ IF recip
                                                                   status ↦ Status.ABORT
                                                               ])}]
                                        ∧ msg' = [msg EXCEPT ![self] = NULL]
                                  ELSE  ∧ IF msg[self].status = Status.SUCCESS
                                             THEN  ∧ state' = [state EXCEPT ![self] = [type ↦
                                                   ∧ IF msg[self] = NULL
                                                        THEN  ∧ msgs' = [recipient ∈ Name
                                                        ELSE  ∧ msgs' = [recipient ∈ Name
                                                   ∧ msg' = [msg EXCEPT ![self] = NULL]
                                             ELSE  ∧ TRUE
                                                   ∧ UNCHANGED ⟨msgs,
                                                               state,
                                                               msg⟩
                       ∧ pc'  = [pc EXCEPT ![self] = "ReachConsensus"]
                       ∧ me' = me

consensusUpdate(self) ≜ ReachConsensus(self) ∨ ReadMessage(self)
                           ∨ ProcessMessage(self)
```

Allow infinite stuttering to prevent deadlock on termination.

$$\begin{aligned} Terminating \triangleq\ & \land \forall\, self \in ProcSet : pc[self] = \text{“Done”} \\ & \land \text{UNCHANGED } vars \end{aligned}$$

$$\begin{aligned} Next \triangleq\ & (\exists\, self \in \text{DOMAIN } Participants : consensusUpdate(self)) \\ & \lor Terminating \end{aligned}$$

$$\begin{aligned} Spec \triangleq\ & \land Init \land \Box[Next]_{vars} \\ & \land \forall\, self \in \text{DOMAIN } Participants : \text{WF}_{vars}(consensusUpdate(self)) \end{aligned}$$

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

END TRANSLATION

$$\begin{aligned} & AllowedMessages \triangleq \\ & \quad [\\ & \qquad turnNumber : Nat, \\ & \qquad votesRequired : 0 \,..\, (NumParticipants - 1), \\ & \qquad status : \{Status.OK\} \\ & \quad] \\ & \quad \cup [\\ & \qquad status : \{Status.ABORT,\ Status.SUCCESS\} \\ & \quad] \end{aligned}$$

$$\begin{aligned} & States \triangleq \{\} \\ & \quad \cup \quad [turnNumber : Nat,\ ourIndex : \text{DOMAIN } Participants,\ type : Range(Types)] \end{aligned}$$

Safety properties

$$\begin{aligned} & TypeOK \triangleq \\ & \quad \text{The following two conditions specify the format of each message and} \\ & \quad \text{participant state.} \\ & \quad \land state \in [\text{DOMAIN } Participants \to States] \\ & \quad \land \forall\, p \in Names : \forall\, m \in msgs[p] : msg \in AllowedMessages \end{aligned}$$

TODO: Get *TurnNumberDoesNotDecrease* and *StaysTerminated*

For some reason, $state[p].turnNumber'$ is not valid

$$\begin{aligned} & TurnNumberDoesNotDecrease \triangleq \\ & \quad \land \forall\, p \in \text{DOMAIN } Participants : state[p].turnNumber' \geq state[p].turnNumber \end{aligned}$$

Once a process has terminated, its state does not change.

$$StaysTerminated \triangleq \forall\, p \in \text{DOMAIN } Participants : (Terminated(state[p]) \Rightarrow (state'[p] = state[p]))$$

Liveness properties

The protocol always terminates consistently across all processes.

TODO: Is this actually feasible, or actually what we want?

For example, perhaps the last person to vote agrees, and sends a message reaching consensus.

Their process terminates in the *SUCCESS* state, but for whatever reason their

commitment was invalid, and the other processes therefore terminate in $FAILURE$.

$ProtocolTerminates \triangleq$

$\lor \land (\forall\, p \in \text{DOMAIN } Participants : \Diamond\Box(state[p].type = Types.SUCCESS))$

$\land \text{TRUE}$ *TODO*: In this case, should we specify that they reach the same turn number?

$\lor (\forall\, p \in \text{DOMAIN } Participants : \Diamond\Box(state[p].type = Types.FAILURE))$

The value of *msg* should eventually always be $NULL$

$MessagesAreRead \triangleq \Diamond\Box(msgs = [p \in \text{DOMAIN } Participants \mapsto \{\text{“Foo”}\}])$

* Modification History

* Last modified *Thu Aug* 15 17:18:02 *PDT* 2019 by *andrewstewart*

* Created *Tue Aug* 06 14:38:11 *MDT* 2019 by *andrewstewart*