———————————— MODULE *ConsensusUpdate* ————————————

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

CONSTANTS

 Names, a set

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

 NULL

ASSUME

 $\land\ Len(Participants) > 1$

$NumParticipants \triangleq Len(Participants)$

$Types \triangleq [$
 $WAITING \mapsto \text{“WAITING”},$
 $SENT \qquad \mapsto \text{“SENT”},$
 $SUCCESS \mapsto \text{“SUCCESS”},$
 $FAILURE \mapsto \text{“FAILURE”}$
$]$

$Status \triangleq [$
 $OK \qquad\quad \mapsto \text{“OK”},$
 $ABORT \qquad \mapsto \text{“ABORT”},$
 $SUCCESS \quad \mapsto \text{“SUCCESS”}$
$]$

$Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$

$Running(state) \triangleq state.type \in \{Types.WAITING,\ Types.SENT\}$

$Terminated(state) \triangleq \neg Running(state)$

--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 $msg = NULL$

define

 Arrays are 1-indexed, while the % operator returns a number between 0 and *NumParticipants*.

 This explains the following slightly complicated expression

$mover(turnNumber) \triangleq 1 + ((turnNumber - 1)\%NumParticipants)$

$safeToSend(state) \quad\triangleq$
  $\land\ state.type = Types.WAITING$
  $\land\ \lor\ state.ourIndex = mover(state.turnNumber)$
   $\lor\ \land\ msg \neq NULL$
    $\land\ msg.status = Status.OK$
    $\land\ state.ourIndex = mover(msg.turnNumber)$

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

end define ;

macro $sendVote(turnNumber, votesRequired)$

begin

assert $votesRequired > 0$ **;**

1

```
state := [
    type ↦ Types.SENT,
    turnNumber ↦ turnNumber,
    ourIndex     ↦ state.ourIndex
] ;
msg := [
    to ↦ target(state.turnNumber),
    turnNumber         ↦ state.turnNumber,
    votesRequired      ↦ votesRequired,
    status             ↦ Status.OK
]
end macro ;


macro returnSuccess()
begin
state := [type ↦ Types.SUCCESS] @@ state ;
msg := [
    to       ↦ target(state.turnNumber),
    status ↦ Status.SUCCESS
]
end macro ;

macro returnFailure(turnNumber)
begin
state := [
    type ↦ Types.FAILURE,
    turnNumber ↦ turnNumber
] @@ state ;
msg := [
    to ↦ target(state.ourIndex + 1),
    status ↦ Status.ABORT
] ;
end macro ;

macro vote(turnNumber, votesRequired)
begin
if votesRequired = 0 then returnSuccess()
 else sendVote(turnNumber, votesRequired)
end if ; end macro ;

macro waitForUpdate(turnNumber)
begin
state := [
    turnNumber ↦ turnNumber,
    type           ↦ Types.WAITING,
    ourIndex     ↦ state.ourIndex
```

```
];
msg := NULL;
end macro ;

macro voteOrreturnFailure(turnNumber, votesRequired)
begin
  If the participant agrees with the allocation, they vote
either vote(turnNumber, votesRequired)
  Otherwise, they return FAILURE
or returnFailure(turnNumber)
end either ; 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]

begin
  Each participant either sends a message if it's currently safe to do so, or else it reads a mes-
  sage 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 safeToSend(state) ∧ msg = NULL then
            either returnFailure(state.turnNumber)   If the commitment is not valid
            or
                if    state.type  = Types.WAITING then vote(state.turnNumber + 1, NumParticipants − 1);
                 elsif state.type = Types.SENT      then returnFailure(state.turnNumber);
                 else assert FALSE
                 end if   ;
            end either ;
          else
            await msg ≠ NULL ∧ msg.to = me ;
            if msg.status = Status.OK then
                If the commitment received 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)*
or if $msg.turnNumber > state.turnNumber$ **then**
  First, update our state based on the incoming message
  if $msg.votesRequired = 0$ **then** *returnSuccess()*
    elsif *safeToSend(state)* **then**
      if $state.type = Types.SENT$ **then** *returnFailure(msg.turnNumber)*
        elsif $state.type = Types.WAITING$ **then** *voteOrreturnFailure(msg.turnNumber + 1, n*
        else assert FALSE ;
        end if ;
      else *waitForUpdate(msg.turnNumber)*
      end if ;
  end if ; **end either** ;
elsif $msg.status = Status.ABORT$ **then** *returnFailure(state.turnNumber)*
elsif $msg.status = Status.SUCCESS$ **then**
  either *returnSuccess()* *If I m ok with the commitment*
  or   *returnFailure(state.turnNumber)* If you sent me an invalid commitment
  end either ;

      end if ;
    end if ;
  end while ;
end process ;

end algorithm ;
BEGIN TRANSLATION
VARIABLES msg, pc

define statement
$mover(turnNumber) \triangleq 1 + ((turnNumber - 1)\% NumParticipants)$
$safeToSend(state) \triangleq$
  $\wedge\ state.type = Types.WAITING$
  $\wedge\ \vee\ state.ourIndex = mover(state.turnNumber)$
    $\vee\ \wedge\ msg \neq NULL$
      $\wedge\ msg.status = Status.OK$
      $\wedge\ state.ourIndex = mover(msg.turnNumber)$
$target(turnNumber) \triangleq Participants[mover(turnNumber)]$

VARIABLES $state$, me

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

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

$Init \triangleq$ Global variables

$\wedge\ msg = NULL$

Process *consensusUpdate*

$\wedge\ state = [self \in \text{DOMAIN } Participants \mapsto \qquad [$
$turnNumber \mapsto 1,$
$ourIndex \mapsto self,$
$type \mapsto Types.WAITING$
$]]$
$\wedge\ me = [self \in \text{DOMAIN } Participants \mapsto Participants[self]]$
$\wedge\ pc\ = [self \in ProcSet \mapsto \text{"ReachConsensus"}]$

$ReachConsensus(self)\ \triangleq\ \wedge\ pc[self] = \text{"ReachConsensus"}$
$\wedge\ \text{IF } Running(state[self])$
$\text{THEN }\ \wedge\ \text{IF } safeToSend(state[self]) \wedge msg = NULL$
$\text{THEN }\ \wedge\ \vee\ \wedge state' = [state \text{ EXCEPT }![self] = \qquad [$
$type \mapsto Types.F$
$turnNumber \mapsto$
$]\ @@ state[self]]$
$\wedge\ msg' = \qquad [$
$to \mapsto target(state'[self].ourIndex + 1),$
$status \mapsto Status.ABORT$
$]$
$\vee\ \wedge \text{IF } state[self].type = Types.WAITING$
$\text{THEN }\ \wedge \text{IF } (NumParticipants - 1) = 0$
$\text{THEN }\ \wedge state' = [state \text{ EXCEPT }![s$
$\wedge\ msg' = \qquad [$
$to \qquad \mapsto target$
$status \mapsto Status$
$]$
$\text{ELSE }\ \wedge Assert((NumParticipants$
$\text{"Failure of assertion}$
$\wedge state' = [state \text{ EXCEPT }![s$

$\wedge\ msg' = \qquad [$
$to \mapsto target(sta$
$turnNumber$
$votesRequired$
$status$
$]$
$\text{ELSE }\ \wedge \text{IF } state[self].type = Types.SENT$
$\text{THEN }\ \wedge state' = [state \text{ EXCEPT }![s$

5

$$\wedge msg' = \quad [\\
\qquad to \mapsto target(sta \\
\qquad status \mapsto Status \\
\qquad] \\
$$

ELSE $\quad \wedge Assert(\textsc{false},$
$\qquad\qquad$ "Failure of assertion
$\qquad \wedge \textsc{unchanged} \ \langle msg,$
$\qquad\qquad\qquad\qquad state\rangle$

ELSE $\quad \wedge msg \neq NULL \wedge msg.to = me[self]$
$\qquad \wedge \textsc{if} \ msg.status = Status.OK$
$\qquad\qquad \textsc{then} \ \ \wedge \vee \wedge state' = [state \ \textsc{except} \ ![self] =$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad typ$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad tur$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad] @@ st$
$\qquad\qquad\qquad\qquad \wedge msg' = \qquad [\\
\qquad\qquad\qquad\qquad\qquad to \mapsto target(state'[self].ourI \\
\qquad\qquad\qquad\qquad\qquad status \mapsto Status.ABORT \\
\qquad\qquad\qquad\qquad]$

$\qquad\qquad\qquad \vee \wedge \textsc{if} \ msg.turnNumber > state[self].turnl$
$\qquad\qquad\qquad\qquad \textsc{then} \ \wedge \textsc{if} \ msg.votesRequired = 0$
$\qquad\qquad\qquad\qquad\qquad\qquad \textsc{then} \ \wedge state' = [state$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge msg' =$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad to$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad sta$
$\qquad\qquad\qquad\qquad\qquad\qquad]$
$\qquad\qquad\qquad\qquad\qquad \textsc{else} \ \wedge \textsc{if} \ safeToSen$
$\qquad\qquad\qquad\qquad\qquad\qquad \textsc{then} \ \wedge I$

$\wedge$ r

ELSE $\wedge$ TRUE
        $\wedge$ UNCHANGED $\langle msg,$
                          $state\rangle$

ELSE $\wedge$ IF $msg.status = Status.ABORT$
        THEN $\wedge$ $state' = [state$ EXCEPT $![self] =$

$\wedge$ $msg' =$ $[$
            $to \mapsto target(state'[se$
            $status \mapsto Status.AB($
                    $]$

ELSE $\wedge$ IF $msg.status = Status.SUCCE$
        THEN $\wedge$ $\vee$ $\wedge$ $state' = [state$
                $\wedge$ $msg' =$
                        to

7

$$sta$$
$$]$$
$$\lor \ \land state' = [state$$

$$\land msg' =$$
$$to$$
$$sta$$
$$]$$
ELSE $\quad \land$ TRUE
$\quad \land$ UNCHANGED $\langle msg$
$$state$$
$$\land pc' = [pc \text{ EXCEPT } ![self] = \text{``ReachConsensus''}]$$
ELSE $\quad \land pc' = [pc \text{ EXCEPT } ![self] = \text{``Done''}]$
$\quad \land$ UNCHANGED $\langle msg,\ state \rangle$
$$\land me' = me$$

$consensusUpdate(self) \ \triangleq \ ReachConsensus(self)$

$Terminating \ \triangleq \ \land \forall self \in ProcSet : pc[self] = \text{``Done''}$
$\qquad\qquad\quad \land$ UNCHANGED $vars$

$Next \ \triangleq \ (\exists self \in \text{DOMAIN } Participants : consensusUpdate(self))$
$\qquad\quad \lor Terminating$

$Spec \ \triangleq \ \land Init \land \Box[Next]_{vars}$
$\qquad\quad \land \forall self \in \text{DOMAIN } Participants : \text{WF}_{vars}(consensusUpdate(self))$

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

END TRANSLATION

$AllowedMessages \ \triangleq$
$[$
$\quad turnNumber : Nat,$
$\quad votesRequired : 0 \mathinner{.\,.} (NumParticipants - 1),$
$\quad to : Names,$
$\quad status : \{Status.OK\}$
$]$
$\cup \{NULL\}$
$\cup [$
$\quad to : Names,$
$\quad status : \{Status.ABORT,\ Status.SUCCESS\}$
$]$

8

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

Safety properties

$TypeOK \triangleq$
   The following two conditions specify the format of each message and
   participant state.
   $\wedge\ state \in [\text{DOMAIN}\ Participants \rightarrow States]$
   $\wedge\ msg \in AllowedMessages$

   $TODO$: Get $TurnNumberDoesNotDecrease$ and $StaysTerminated$
   For some reason, $state[p].turnNumber'$ is not valid
$TurnNumberDoesNotDecrease \triangleq$
   $\wedge\ \forall\, p \in \text{DOMAIN}\ Participants : state[p].turnNumber' \geq state[p].turnNumber$

   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 $FAILURE$.
$ProtocolTerminates \triangleq$
     $\vee\ \wedge (\forall\, p \in \text{DOMAIN}\ Participants : \Diamond\Box(state[p].type = Types.SUCCESS))$
        $\wedge\ \text{TRUE}$ $TODO$: In this case, should we specify that they reach the same turn number?
     $\vee (\forall\, p \in \text{DOMAIN}\ Participants : \Diamond\Box(state[p].type = Types.FAILURE))$

   The value of msg should eventually always be $NULL$
$MessagesAreRead \triangleq \Diamond\Box(msg = NULL)$

\ * Modification History
\ * Last modified *Mon Aug* 12 22:26:37 *MDT* 2019 by *andrewstewart*
\ * Created *Tue Aug* 06 14:38:11 *MDT* 2019 by *andrewstewart*