MODULE *Pactus*

The specification of the *Pactus* consensus algorithm based on Practical *Byzantine* Fault Tolerant.
For more information check here: https://pactus.org/learn/consensus/protocol/

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

CONSTANT

The total number of faulty nodes

NumFaulty,

The maximum number of round per height.
this is to restrict the allowed behaviours that *TLC* scans through.

MaxRound

ASSUME

$\land\ NumFaulty \geq 1$

VARIABLES

log,
states

Total number of replicas that is $3f + 1$ where f is number of faulty nodes.

$Replicas \triangleq (3 * NumFaulty) + 1$

2/3 of total replicas that is $2f + 1$

$QuorumCnt \triangleq (2 * NumFaulty) + 1$

1/3 of total replicas that is $f + 1$

$OneThird \triangleq NumFaulty + 1$

A tuple with all variables in the spec (for ease of use in temporal conditions)

$vars \triangleq \langle states, log \rangle$

Helper functions

Fetch a subset of messages in the network based on the *params* filter.

$SubsetOfMsgs(params) \triangleq$
$\quad \{msg \in log : \forall field \in \text{DOMAIN } params : msg[field] = params[field]\}$

IsProposer checks if the replica is the proposer for this round
To simplify, we assume the proposer always starts with the first replica
and moves to the next by the change-proposer phase.

$IsProposer(index) \triangleq$
$\quad states[index].round\%Replicas = index$

HasPrepareQuorum checks if there is a quorum of the *PREPARE* votes in each round.

$HasPrepareQuorum(index) \triangleq$
$\quad Cardinality(SubsetOfMsgs([$
$\qquad type \mapsto$ "PREPARE",
$\qquad height \mapsto states[index].height,$
$\qquad round \mapsto states[index].round])) \geq QuorumCnt$

HasPrecommitQuorum checks if there is a quorum of the *PRECOMMIT* votes in each round.

$$
\begin{array}{l}
HasPrecommitQuorum(index) \triangleq \\
\quad Cardinality(SubsetOfMsgs([\\
\qquad type \mapsto \text{“PRECOMMIT”}, \\
\qquad height \mapsto states[index].height, \\
\qquad round \mapsto states[index].round])) \geq QuorumCnt
\end{array}
$$

HasChangeProposerQuorum checks if there is a quorum of the CHANGE-PROPOSER votes in each round.

$$
\begin{array}{l}
HasChangeProposerQuorum(index) \triangleq \\
\quad Cardinality(SubsetOfMsgs([\\
\qquad type \mapsto \text{“CHANGE-PROPOSER”}, \\
\qquad height \mapsto states[index].height, \\
\qquad round \mapsto states[index].round])) \geq QuorumCnt
\end{array}
$$

$$
\begin{array}{l}
HasOneThirdOfChangeProposer(index) \triangleq \\
\quad Cardinality(SubsetOfMsgs([\\
\qquad type \mapsto \text{“CHANGE-PROPOSER”}, \\
\qquad height \mapsto states[index].height, \\
\qquad round \mapsto states[index].round])) \geq OneThird
\end{array}
$$

$$
\begin{array}{l}
GetProposal(height,\ round) \triangleq \\
\quad SubsetOfMsgs([type \mapsto \text{“PROPOSAL”},\ height \mapsto height,\ round \mapsto round])
\end{array}
$$

$$
\begin{array}{l}
HasProposal(height,\ round) \triangleq \\
\quad Cardinality(GetProposal(height,\ round)) > 0
\end{array}
$$

$$
\begin{array}{l}
IsCommitted(height) \triangleq \\
\quad Cardinality(SubsetOfMsgs([type \mapsto \text{“BLOCK-ANNOUNCE”},\ height \mapsto height])) > 0
\end{array}
$$

$$
\begin{array}{l}
HasVoted(index,\ type) \triangleq \\
\quad Cardinality(SubsetOfMsgs([\\
\qquad type \mapsto type, \\
\qquad height \mapsto states[index].height, \\
\qquad round \mapsto states[index].round, \\
\qquad index \mapsto index])) > 0
\end{array}
$$

Network functions

SendMsg broadcasts the message iff the current height is not committed yet.

$$
\begin{array}{l}
SendMsg(msg) \triangleq \\
\quad \text{IF } \neg IsCommitted(msg.height) \\
\quad \text{THEN } log' = log \cup \{msg\} \\
\quad \text{ELSE } log' = log
\end{array}
$$

SendProposal is used to broadcast the *PROPOSAL* into the network.

$$
SendProposal(index) \triangleq
$$

$SendMsg([$
$type \mapsto$ "PROPOSAL",
$height \mapsto states[index].height,$
$round \mapsto states[index].round,$
$index \mapsto index])$

SendPrepareVote is used to broadcast *PREPARE* votes into the network.

$SendPrepareVote(index) \triangleq$
$SendMsg([$
$type \mapsto$ "PREPARE",
$height \mapsto states[index].height,$
$round \mapsto states[index].round,$
$index \mapsto index])$

SendPrecommitVote is used to broadcast *PRECOMMIT* votes into the network.

$SendPrecommitVote(index) \triangleq$
$SendMsg([$
$type \mapsto$ "PRECOMMIT",
$height \mapsto states[index].height,$
$round \mapsto states[index].round,$
$index \mapsto index])$

SendChangeProposerRequest is used to broadcast CHANGE-PROPOSER votes into the network.

$SendChangeProposerRequest(index) \triangleq$
$SendMsg([$
$type \mapsto$ "CHANGE-PROPOSER",
$height \mapsto states[index].height,$
$round \mapsto states[index].round,$
$index \mapsto index])$

AnnounceBlock announces the block for the current height and clears the logs.

$AnnounceBlock(index) \triangleq$
$log' = \{msg \in log : (msg.type =$ "BLOCK-ANNOUNCE"$) \lor msg.height > states[index].height\} \cup \{[$
$type \mapsto$ "BLOCK-ANNOUNCE",
$height \mapsto states[index].height,$
$round \mapsto states[index].round,$
$index \mapsto -1]\}$

$IsFaulty(index) \triangleq index \geq 3 * NumFaulty$

States functions

NewHeight state

$NewHeight(index) \triangleq$
$\land states[index].name =$ "new-height"
$\land \neg IsFaulty(index)$
$\land states' = [states$ EXCEPT

```
        ![index].name = "propose",
        ![index].height = states[index].height + 1,
        ![index].round = 0]
    ∧ UNCHANGED ⟨log⟩

 Propose state
Propose(index) ≜
    ∧  states[index].name = "propose"
    ∧  ¬IsFaulty(index)
    ∧  IF IsProposer(index)
       THEN SendProposal(index)
       ELSE log' = log
    ∧  states' = [states EXCEPT
       ![index].name = "prepare"]

 Prepare state
Prepare(index) ≜
    ∧  states[index].name = "prepare"
    ∧  ¬IsFaulty(index)
    ∧  ¬HasOneThirdOfChangeProposer(index)  This check is optional
    ∧  HasProposal(states[index].height, states[index].round)
    ∧  SendPrepareVote(index)
    ∧  IF  ∧ HasPrepareQuorum(index)
       THEN states' = [states EXCEPT ![index].name = "precommit"]
       ELSE states' = states

 Precommit state
Precommit(index) ≜
    ∧ states[index].name = "precommit"
    ∧ ¬IsFaulty(index)
    ∧ ¬HasOneThirdOfChangeProposer(index)  This check is optional
    ∧ SendPrecommitVote(index)
    ∧ IF  ∧ HasPrecommitQuorum(index)
          ∧ HasVoted(index, "PRECOMMIT")
      THEN states' = [states EXCEPT ![index].name = "commit"]
      ELSE states' = states

Timeout(index) ≜
    ∧
        ∨ states[index].name = "prepare"
        ∨ states[index].name = "precommit"
    ∧ ¬IsFaulty(index)
    ∧ (states[index].round < MaxRound)
```

```
    ∧ SendChangeProposerRequest(index)
    ∧ states' = [states EXCEPT
        ![index].name = "change-proposer"]

 Commit state
Commit(index) ≜
    ∧ states[index].name = "commit"
    ∧ ¬IsFaulty(index)
    ∧ AnnounceBlock(index)  this clear the logs
    ∧ states' = [states EXCEPT
        ![index].name = "new-height"]


 ChangeProposer state
ChangeProposer(index) ≜
    ∧ states[index].name = "change-proposer"
    ∧ ¬IsFaulty(index)
    ∧ IF HasChangeProposerQuorum(index)
      THEN states' = [states EXCEPT
             ![index].name = "propose",
             ![index].round = states[index].round + 1]
      ELSE states' = states
    ∧ UNCHANGED ⟨log⟩

 Sync checks the log for the committed blocks at the current height.
 If such a block exists, it commits and moves to the next height.
Sync(index) ≜
    LET
      blocks ≜ SubsetOfMsgs([type ↦ "BLOCK-ANNOUNCE", height ↦ states[index].height])
    IN
      ∧ Cardinality(blocks) > 0
      ∧ states' = [states EXCEPT
          ![index].name = "propose",
          ![index].height = states[index].height + 1,
          ![index].round = 0]
      ∧ log' = log

──────────────────────────────────────────────────────────────────────

Init ≜
    ∧ log = {}
    ∧ states = [index ∈ 0 .. Replicas − 1 ↦ [
        name      ↦ "new-height",
        height    ↦ 0,
        round     ↦ 0]]

Next ≜
```

$\exists\, index \in 0\,..\,Replicas - 1 :$
$\lor\ Sync(index)$
$\lor\ NewHeight(index)$
$\lor\ Propose(index)$
$\lor\ Prepare(index)$
$\lor\ Precommit(index)$
$\lor\ Timeout(index)$
$\lor\ Commit(index)$
$\lor\ ChangeProposer(index)$

$Spec \triangleq$
$Init \land \Box[Next]_{vars}$

TypeOK is the type-correctness invariant.

$TypeOK \triangleq$
$\land \quad \forall\, index \in 0\,..\,Replicas - 1 :$
$\land\ states[index].name \in \{$"new-height", "propose", "prepare",
"precommit", "commit", "change-proposer"$\}$
$\land\ \neg IsCommitted(states[index].height) \Rightarrow$
$\land\ states[index].name =$ "new-height" $\land\ states[index].height > 1 \Rightarrow$
$IsCommitted(states[index].height - 1)$
$\land\ states[index].name =$ "propose" $\land\ states[index].round > 0 \Rightarrow$
$\land\ Cardinality(SubsetOfMsgs([$
$type \quad \mapsto$ "CHANGE-PROPOSER",
$height \mapsto states[index].height,$
$round \mapsto states[index].round - 1])) \geq QuorumCnt$
$\land\ states[index].name =$ "precommit" $\Rightarrow$
$\land\ HasPrepareQuorum(index)$
$\land\ HasProposal(states[index].height,\ states[index].round)$
$\land\ states[index].name =$ "commit" $\Rightarrow$
$\land\ HasPrepareQuorum(index)$
$\land\ HasPrecommitQuorum(index)$
$\land\ HasProposal(states[index].height,\ states[index].round)$
$\land\ \forall\, round \in 0\,..\,states[index].round :$
Not more than one proposal per round
$\land\ Cardinality(GetProposal(states[index].height,\ round)) \leq 1$