MODULE *Pactus*

*The specification of the Pactus consensus algorithm :*
https://docs.pactus.org/protocol/consensus/protocol/

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

CONSTANT

*The maximum number of rounds, limiting the range of behaviors evaluated by TLC.*
*MaxRound*,
*The maximum number of change − proposer(CP)rounds, limiting the range of behaviors evaluated by TLC.*
*MaxCPRound*,
*The total number of nodes in the network, denoted as N in the protocol.*
*N*,
*The maximum number of faulty nodes in the network, denoted as F in the protocol.*
*F*,
*The indices of faulty nodes.*
*FaultyNodes*

VARIABLES

*The set of messages received by the network.*
*network*,
*The set of messages delivered to each replica.*
*logs*,
*The state of each replica in the consensus protocol.*
*states*

*Helper expressions for common values.*
$ThreeFPlusOne \triangleq (3 * F) + 1$
$TwoFPlusOne \triangleq (2 * F) + 1$
$OneFPlusOne \triangleq (1 * F) + 1$

*A tuple containing all variables in the spec for ease of use in temporal conditions.*
$vars \triangleq \langle network, logs, states \rangle$

ASSUME

*Ensure the number of nodes is sufficient to tolerate the specified number of faults.*
$\land N \geq ThreeFPlusOne$
*Ensure that FaultyNodes is a valid subset of node indices.*
$\land FaultyNodes \subseteq 0 \,..\, N - 1$

*Helper functions*

*Check if the replica is the proposer for this round.*
*The proposer starts with the first replica and moves to the next in the change − proposer phase.*
$IsProposer(index) \triangleq$
$states[index].round\%N = index$

*Check if a node is faulty.*

$IsFaulty(index) \triangleq index \in FaultyNodes$

*Returns a subset of bag where each element matches all criteria specified in params*

$SubsetOfMsgs(bag, params) \triangleq$
$\{i \in bag : \forall field \in \text{DOMAIN } params : i[field] = params[field]\}$

*Check if the node has received $3f + 1$ PRECOMMIT votes for a proposal in the current round.*

$HasPreCommitAbsolute(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "PRECOMMIT",
$round \mapsto states[index].round])) \geq ThreeFPlusOne$

*Check if the node has received $2f + 1$ PRECOMMIT votes for a proposal in the current round.*

$HasPreCommitQuorum(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "PRECOMMIT",
$round \mapsto states[index].round])) \geq TwoFPlusOne$

$CPHasPreVotesQuorum(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:PRE-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round])) \geq TwoFPlusOne$

$CPHasPreVotesQuorumForYes(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:PRE-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round,$
$cp\_val \mapsto 1])) \geq TwoFPlusOne$

$CPHasPreVotesQuorumForNo(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:PRE-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round,$
$cp\_val \mapsto 0])) \geq TwoFPlusOne$

$CPHasPreVotesMinorityForNo(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:PRE-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round,$
$cp\_val \mapsto 0])) \geq OneFPlusOne$

$CPHasPreVotesMinorityForYes(index) \triangleq$

$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:PRE-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round,$
$cp\_val \mapsto 1])) \geq OneFPlusOne$

$CPHasPreVotesForYesAndNo(index) \triangleq$
$\land Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:PRE-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round,$
$cp\_val \mapsto 0])) \geq 1$
$\land Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:PRE-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round,$
$cp\_val \mapsto 1])) \geq 1$

$CPHasOneMainVotesNoInPrvRound(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:MAIN-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round - 1,$
$cp\_val \mapsto 0])) > 0$

$CPHasOneMainVotesYesInPrvRound(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:MAIN-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round - 1,$
$cp\_val \mapsto 1])) > 0$

$CPAllMainVotesAbstainInPrvRound(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:MAIN-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round - 1,$
$cp\_val \mapsto 2])) \geq TwoFPlusOne$

$CPOneFPlusOneMainVotesAbstainInPrvRound(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:MAIN-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round - 1,$
$cp\_val \mapsto 2])) \geq OneFPlusOne$

$CPHasMainVotesQuorum(index) \triangleq$

$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "CP:MAIN-VOTE",
$\quad round \mapsto states[index].round,$
$\quad cp\_round \mapsto states[index].cp\_round])) \geq TwoFPlusOne$

$CPHasMainVotesQuorumForNo(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "CP:MAIN-VOTE",
$\quad round \mapsto states[index].round,$
$\quad cp\_round \mapsto states[index].cp\_round,$
$\quad cp\_val \mapsto 0])) \geq TwoFPlusOne$

$CPHasMainVotesQuorumForYes(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "CP:MAIN-VOTE",
$\quad round \mapsto states[index].round,$
$\quad cp\_round \mapsto states[index].cp\_round,$
$\quad cp\_val \mapsto 1])) \geq TwoFPlusOne$

$CPHasDecideVotesForNo(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "CP:DECIDED",
$\quad round \mapsto states[index].round,$
$\quad cp\_val \mapsto 0])) > 0$

$CPHasDecideVotesForYes(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "CP:DECIDED",
$\quad round \mapsto states[index].round,$
$\quad cp\_val \mapsto 1])) > 0$

*Check if the node has received a proposal in the current round.*
$HasProposal(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "PROPOSAL",
$\quad round \mapsto states[index].round])) > 0$

$HasPrecommited(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "PRECOMMIT",
$\quad round \mapsto states[index].round,$
$\quad index \mapsto index])) = 1$

*Check if the node has received a block announce message in the current round.*
$HasBlockAnnounce(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$

```
        type      ↦ "BLOCK-ANNOUNCE",
        round     ↦ states[index].round])) > 0

 Check if the block is committed.
 A block is considered committed if a supermajority of non − faulty replicas announce the same block.
IsCommitted ≜
    LET subset ≜ SubsetOfMsgs(network, [type ↦ "BLOCK-ANNOUNCE"])
    IN  ∧ Cardinality(subset) ≥ TwoFPlusOne
        ∧ ∀ m1, m2 ∈ subset : m1.round = m2.round
```

---

```
 Network functions

 Simulate a replica sending a message by appending it to the network
 The message is delivered to the sender s log immediately.
SendMsg(msg) ≜
    IF msg.cp_round < MaxCPRound THEN
        ∧ network′   = network ∪ {msg}
        ∧ logs′ = [logs EXCEPT ![msg.index] = logs[msg.index] ∪ {msg}]
     ELSE
        UNCHANGED ⟨network, logs⟩

 Deliver a message to the specified replica's log.
DeliverMsg(index) ≜
    LET undeliveredMsgs ≜ network \ logs[index]
    IN  IF Cardinality(undeliveredMsgs) = 0 THEN
          UNCHANGED ⟨vars⟩
     ELSE
        LET msg ≜ CHOOSE x ∈ undeliveredMsgs : TRUE
        IN
            ∧ logs′ = [logs EXCEPT ![index] = logs[index] ∪ {msg}]
            ∧ UNCHANGED ⟨states, network⟩

 Broadcast a PROPOSAL message into the network.
SendProposal(index) ≜
    SendMsg([
        type      ↦ "PROPOSAL",
        round     ↦ states[index].round,
        index     ↦ index,
        cp_round ↦ 0,
        cp_val    ↦ 0])

 Broadcast PRECOMMIT votes into the network.
SendPreCommitVote(index) ≜
    SendMsg([
        type      ↦ "PRECOMMIT",
        round     ↦ states[index].round,
```

```
        index      ↦ index,
        cp_round   ↦ 0,
        cp_val     ↦ 0])

  Broadcast CP:PRE-VOTE votes into the network.
SendCPPreVote(index, cp_val) ≜
    SendMsg([
        type       ↦ "CP:PRE-VOTE",
        round      ↦ states[index].round,
        index      ↦ index,
        cp_round   ↦ states[index].cp_round,
        cp_val     ↦ cp_val])

  Broadcast CP:MAIN-VOTE votes into the network.
SendCPMainVote(index, cp_val) ≜
    SendMsg([
        type       ↦ "CP:MAIN-VOTE",
        round      ↦ states[index].round,
        index      ↦ index,
        cp_round   ↦ states[index].cp_round,
        cp_val     ↦ cp_val])

  Broadcast CP:DECIDED votes into the network.
SendCPDecideVote(index, cp_val) ≜
    SendMsg([
        type       ↦ "CP:DECIDED",
        round      ↦ states[index].round,
        cp_round   ↦ states[index].cp_round,
        index      ↦ 0,    reduce the model size
        cp_val     ↦ cp_val])

  Broadcast BLOCK-ANNOUNCE messages into the network.
AnnounceBlock(index) ≜
    SendMsg([
        type       ↦ "BLOCK-ANNOUNCE",
        round      ↦ states[index].round,
        index      ↦ index,
        cp_round   ↦ 0,
        cp_val     ↦ 0])

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

State transition functions

  Transition to the propose state.
Propose(index) ≜
    ∧  ¬IsFaulty(index)
    ∧  states[index].name = "propose"
```

```
    ∧
        IF IsProposer(index) THEN
            SendProposal(index)
        ELSE
            UNCHANGED ⟨logs, network⟩
    ∧ states′ = [states EXCEPT ![index].name = "precommit"]

 Transition to the precommit state.
PreCommit(index) ≜
    ∧ ¬IsFaulty(index)
    ∧ states[index].name = "precommit"
    ∧ HasProposal(index)
    ∧ SendPreCommitVote(index)
    ∧ states′ = states


 Transition to the fast commit state.
FastCommit(index) ≜
    ∧ ¬IsFaulty(index)
    ∧ states[index].name ≠ "commit"  to prevent shuttering
    ∧ HasPreCommitAbsolute(index)
    ∧ states′ = [states EXCEPT ![index].name = "commit"]
    ∧ UNCHANGED ⟨network, logs⟩

 Transition to the commit state.
Commit(index) ≜
    ∧ ¬IsFaulty(index)
    ∧ states[index].name = "commit"
    ∧ HasProposal(index)
    ∧ HasPreCommitQuorum(index)
    ∧ AnnounceBlock(index)
    ∧ UNCHANGED ⟨states⟩

 Transition for timeout: a non-faulty replica changes the proposer if its timer expires.
Timeout(index) ≜
    ∧   ¬IsFaulty(index)
    ∧   states[index].name = "precommit"
    ∧
         To limit the the behaviours.
        ∨ states[index].round < MaxRound
        ∨ HasPreCommitQuorum(index)
    ∧ states′ = [states EXCEPT ![index].name = "cp:pre-vote"]
    ∧ UNCHANGED ⟨network, logs⟩

 Transition to the CP pre-vote state.
CPPreVote(index) ≜
    ∧ ¬IsFaulty(index)
```

```
    ∧ states[index].name = "cp:pre-vote"
    ∧
        IF states[index].cp_round = 0 THEN
            IF HasPreCommitQuorum(index) THEN
                ∧ SendCPPreVote(index, 0)
                ∧ states' = [states EXCEPT ![index].name = "cp:main-vote"]
            ELSE IF ¬HasPrecommited(index) THEN
                ∧ SendCPPreVote(index, 1)
                ∧ states' = [states EXCEPT ![index].name = "cp:main-vote"]
            ELSE IF ∨ CPHasPreVotesMinorityForYes(index)
                    ∨ Cardinality(
                            SubsetOfMsgs(logs[index], [type ↦ "PRECOMMIT", round ↦ states[index].round
                            SubsetOfMsgs(logs[index], [type ↦ "CP:PRE-VOTE", round ↦ states[index].roun
                        ) ≥ TwoFPlusOne THEN
                ∧ SendCPPreVote(index, 1)
                ∧ states' = [states EXCEPT ![index].name = "cp:main-vote"]
            ELSE
                ∧ UNCHANGED ⟨vars⟩
        ELSE
            ∧
                ∨
                    ∧ CPHasOneMainVotesNoInPrvRound(index)
                    ∧ SendCPPreVote(index, 0)
                ∨
                    ∧ CPHasOneMainVotesYesInPrvRound(index)
                    ∧ SendCPPreVote(index, 1)
                ∨
                    ∧ CPAllMainVotesAbstainInPrvRound(index)
                    ∧ SendCPPreVote(index, 0) biased to zero
            ∧ states' = [states EXCEPT ![index].name = "cp:main-vote"]

Transition to the CP main-vote state.
CPMainVote(index) ≜
    ∧ ¬IsFaulty(index)
    ∧ states[index].name = "cp:main-vote"
    ∧ CPHasPreVotesQuorum(index)
    ∧
        ∨
            all votes for 0
            ∧ CPHasPreVotesQuorumForNo(index)
            ∧ CPHasPreVotesMinorityForNo(index) To reduce the behaviours.
            ∧ states' = [states EXCEPT ![index].name = "commit"]
            ∧ UNCHANGED ⟨network, logs⟩
        ∨
            all votes for 1
```

```
            ∧ CPHasPreVotesQuorumForYes(index)
            ∧ SendCPMainVote(index, 1)
            ∧ states' = [states EXCEPT ![index].name = "cp:decide"]
        ∨
            Abstain vote
            ∧ CPHasPreVotesForYesAndNo(index)
            ∧ SendCPMainVote(index, 2) Abstain
            ∧ states' = [states EXCEPT ![index].name = "cp:decide"]

Transition to the CP decide state.
CPDecide(index) ≜
    ∧ ¬IsFaulty(index)
    ∧ states[index].name = "cp:decide"
    ∧  CPHasMainVotesQuorum(index)
    ∧
        IF CPHasMainVotesQuorumForYes(index) THEN
            ∧ states' = [states EXCEPT ![index].name = "propose",
                                       ![index].round = states[index].round + 1]
            ∧ SendCPDecideVote(index, 1)

        ELSE
            ∧ states' = [states EXCEPT ![index].name = "cp:pre-vote",
                                       ![index].cp_round = states[index].cp_round + 1]
            ∧ UNCHANGED ⟨network, logs⟩

Transition for strong termination of Change-Proposer phase.
CPStrongTerminate(index) ≜
    ∧ ¬IsFaulty(index)
    ∧
        ∨ states[index].name = "cp:pre-vote"
        ∨ states[index].name = "cp:main-vote"
        ∨ states[index].name = "cp:decide"
    ∧
        IF HasBlockAnnounce(index) THEN
            ∧ states' = [states EXCEPT ![index].name = "commit"]

        To limit the the behaviours.
        ELSE IF ∧ states[index].cp_round = MaxCPRound
                ∧ CPOneFPlusOneMainVotesAbstainInPrvRound(index) THEN
            ∧ states' = [states EXCEPT ![index].name = "commit"]

        ELSE IF CPHasDecideVotesForYes(index) THEN
            ∧ states' = [states EXCEPT ![index].name = "propose",
                                       ![index].round = states[index].round + 1]
        ELSE
            ∧ states' = states
```

$\land$ UNCHANGED $\langle network,\ logs \rangle$

---

Initial state

$$
\begin{array}{l}
Init \triangleq \\
\quad \land\ network = \{\} \\
\quad \land\ logs = [index \in 0\,..\,N-1 \mapsto \{\}] \\
\quad \land\ states = [index \in 0\,..\,N-1 \mapsto [ \\
\qquad name \quad \mapsto \text{“propose”}, \\
\qquad round \quad \mapsto 0, \\
\qquad cp\_round \quad \mapsto 0]]
\end{array}
$$

State transition relation

$$
\begin{array}{l}
Next \triangleq \\
\quad \exists\, index \in 0\,..\,N-1 : \\
\qquad \lor\ Propose(index) \\
\qquad \lor\ PreCommit(index) \\
\qquad \lor\ Timeout(index) \\
\qquad \lor\ Commit(index) \\
\qquad \lor\ FastCommit(index) \\
\qquad \lor\ CPPreVote(index) \\
\qquad \lor\ CPMainVote(index) \\
\qquad \lor\ CPDecide(index) \\
\qquad \lor\ CPStrongTerminate(index) \\
\qquad \lor\ DeliverMsg(index)
\end{array}
$$

Specification

$$
\begin{array}{l}
Spec \triangleq \\
\quad Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)
\end{array}
$$

Success: All non-faulty nodes eventually commit.

$$Success \triangleq \Diamond(IsCommitted)$$

*TypeOK* is the type-correctness invariant.

$$
\begin{array}{l}
TypeOK \triangleq \\
\quad \land \quad \forall\, index \in 0\,..\,N-1 : \\
\qquad \land\ states[index].round \leq MaxRound \\
\qquad \land\ states[index].cp\_round \leq MaxCPRound \\
\qquad \land\ states[index].name = \text{“propose”} \land states[index].round > 0 \Rightarrow \\
\qquad\quad \land\ Cardinality(SubsetOfMsgs(network, [ \\
\qquad\qquad type \quad \mapsto \text{“CP:DECIDED”}, \\
\qquad\qquad round \quad \mapsto states[index].round - 1, \\
\qquad\qquad cp\_val \quad \mapsto 1])) = 1 \\
\qquad\quad \land\ Cardinality(SubsetOfMsgs(network, [ \\
\qquad\qquad type \quad \mapsto \text{“BLOCK-ANNOUNCE”},
\end{array}
$$

```
        round     ↦ states[index].round − 1])) = 0
∧ states[index].name = "commit" ⇒
    ∧ Cardinality(SubsetOfMsgs(network, [
        type      ↦ "PRECOMMIT",
        round     ↦ states[index].round])) ≥ TwoFPlusOne
    ∧ Cardinality(SubsetOfMsgs(network, [
        type      ↦ "PROPOSAL",
        round     ↦ states[index].round])) = 1
    ∧ LET subset ≜ SubsetOfMsgs(network, [type ↦ "BLOCK-ANNOUNCE"])
      IN  ∧ ∀ m1, m2 ∈ subset : m1.round = m2.round
```