MODULE *Pactus*

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

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

CONSTANT

The maximum number of height.
this is to restrict the allowed behaviours that TLC scans through.
MaxHeight,
The maximum number of round per height.
this is to restrict the allowed behaviours that TLC scans through.
MaxRound,
The maximum number of cp − round per height.
this is to restrict the allowed behaviours that TLC scans through.
MaxCPRound,
The total number of faulty nodes
NumFaulty,
The index of faulty nodes
FaultyNodes

VARIABLES

log is a set of received messages in the system.
log,
states represents the state of each replica in the consensus protocol.
states

Total number of replicas, *which is* $3f + 1$, where f is the number of faulty nodes.
$Replicas \triangleq (3 * NumFaulty) + 1$
Quorum is $2/3+$ of total replicas that is $2f + 1$
$Quorum \triangleq (2 * NumFaulty) + 1$
OneThird is $1/3+$ 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)
$vars \triangleq \langle states, log \rangle$

ASSUME

$\land\ NumFaulty \geq 1$
$\land\ FaultyNodes \subseteq 0\,..\,Replicas - 1$

Helper functions

Fetch a subset of messages in the network based on the *params* filter.
$SubsetOfMsgs(params) \triangleq$
$\{msg \in log : \forall\, field \in \text{DOMAIN } params : msg[field] = params[field]\}$

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.

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

Helper function to check if a node is faulty or not.

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

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

$$HasPrepareQuorum(index) \triangleq$$
$$\quad Cardinality(SubsetOfMsgs([$$
$$\qquad type \mapsto \text{“PREPARE”},$$
$$\qquad height \mapsto states[index].height,$$
$$\qquad round \mapsto states[index].round,$$
$$\qquad cp\_round \mapsto 0])) \geq Quorum$$

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

$$HasPrecommitQuorum(index) \triangleq$$
$$\quad Cardinality(SubsetOfMsgs([$$
$$\qquad type \mapsto \text{“PRECOMMIT”},$$
$$\qquad height \mapsto states[index].height,$$
$$\qquad round \mapsto states[index].round,$$
$$\qquad cp\_round \mapsto 0])) \geq Quorum$$

$$CPHasPreVotesQuorum(index) \triangleq$$
$$\quad Cardinality(SubsetOfMsgs([$$
$$\qquad type \mapsto \text{“CP:PRE-VOTE”},$$
$$\qquad height \mapsto states[index].height,$$
$$\qquad round \mapsto states[index].round,$$
$$\qquad cp\_round \mapsto states[index].cp\_round])) \geq Quorum$$

$$CPHasPreVotesQuorumForOne(index) \triangleq$$
$$\quad Cardinality(SubsetOfMsgs([$$
$$\qquad type \mapsto \text{“CP:PRE-VOTE”},$$
$$\qquad height \mapsto states[index].height,$$
$$\qquad round \mapsto states[index].round,$$
$$\qquad cp\_round \mapsto states[index].cp\_round,$$
$$\qquad cp\_val \mapsto 1])) \geq Quorum$$

$$CPHasPreVotesQuorumForZero(index) \triangleq$$
$$\quad Cardinality(SubsetOfMsgs([$$
$$\qquad type \mapsto \text{“CP:PRE-VOTE”},$$
$$\qquad height \mapsto states[index].height,$$
$$\qquad round \mapsto states[index].round,$$

```
        cp_round ↦ states[index].cp_round,
        cp_val   ↦ 0])) ≥ Quorum

CPHasPreVotesForZeroAndOne(index) ≜
    ∧ Cardinality(SubsetOfMsgs([
        type     ↦ "CP:PRE-VOTE",
        height   ↦ states[index].height,
        round    ↦ states[index].round,
        cp_round ↦ states[index].cp_round,
        cp_val   ↦ 0])) ≥ 1
    ∧ Cardinality(SubsetOfMsgs([
        type     ↦ "CP:PRE-VOTE",
        height   ↦ states[index].height,
        round    ↦ states[index].round,
        cp_round ↦ states[index].cp_round,
        cp_val   ↦ 1])) ≥ 1

CPHasOneMainVotesZeroInPrvRound(index) ≜
    Cardinality(SubsetOfMsgs([
        type     ↦ "CP:MAIN-VOTE",
        height   ↦ states[index].height,
        round    ↦ states[index].round,
        cp_round ↦ states[index].cp_round − 1,
        cp_val   ↦ 0])) > 0

CPHasOneMainVotesOneInPrvRound(index) ≜
    Cardinality(SubsetOfMsgs([
        type     ↦ "CP:MAIN-VOTE",
        height   ↦ states[index].height,
        round    ↦ states[index].round,
        cp_round ↦ states[index].cp_round − 1,
        cp_val   ↦ 1])) > 0

CPAllMainVotesAbstainInPrvRound(index) ≜
    Cardinality(SubsetOfMsgs([
        type     ↦ "CP:MAIN-VOTE",
        height   ↦ states[index].height,
        round    ↦ states[index].round,
        cp_round ↦ states[index].cp_round − 1,
        cp_val   ↦ 2])) ≥ Quorum


CPHasMainVotesQuorum(index) ≜
    Cardinality(SubsetOfMsgs([
        type     ↦ "CP:MAIN-VOTE",
        height   ↦ states[index].height,
        round    ↦ states[index].round,
```

```
        cp_round ↦ states[index].cp_round])) ≥ Quorum

CPHasMainVotesQuorumForOne(index) ≜
    Cardinality(SubsetOfMsgs([
        type      ↦ "CP:MAIN-VOTE",
        height    ↦ states[index].height,
        round     ↦ states[index].round,
        cp_round  ↦ states[index].cp_round,
        cp_val    ↦ 1])) ≥ Quorum

CPHasMainVotesQuorumForZero(index) ≜
    Cardinality(SubsetOfMsgs([
        type      ↦ "CP:MAIN-VOTE",
        height    ↦ states[index].height,
        round     ↦ states[index].round,
        cp_round  ↦ states[index].cp_round,
        cp_val    ↦ 0])) ≥ Quorum

GetProposal(height, round) ≜
    SubsetOfMsgs([type ↦ "PROPOSAL", height ↦ height, round ↦ round])

HasProposal(index) ≜
    Cardinality(GetProposal(states[index].height, states[index].round)) > 0

HasBlockAnnounce(index) ≜
    Cardinality(SubsetOfMsgs([
        type      ↦ "BLOCK-ANNOUNCE",
        height    ↦ states[index].height,
        round     ↦ states[index].round,
        cp_round  ↦ 0,
        cp_val    ↦ 0])) ≥ 1
```

Helper function to check if the block is committed or not.
A block is considered committed iff supermajority of non-faulty replicas announce the same block.

```
IsCommitted(height) ≜
    LET subset ≜ SubsetOfMsgs([
            type      ↦ "BLOCK-ANNOUNCE",
            height    ↦ height,
            cp_round  ↦ 0,
            cp_val    ↦ 0])
    IN  ∧ Cardinality(subset) ≥ Quorum
        ∧ ∀ m1, m2 ∈ subset : m1.round = m2.round
```

Network functions

SendMsg simulates a replica sending a message by appending it to the log

$SendMsg(msg) \triangleq$
    $log' = log \cup msg$

SendProposal is used to broadcast the PROPOSAL into the network.
$SendProposal(index) \triangleq$
    $SendMsg(\{[$
        $type \mapsto$ "PROPOSAL",
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round,$
        $index \mapsto index,$
        $cp\_round \mapsto 0,$
        $cp\_val \mapsto 0]\})$

SendPrepareVote is used to broadcast PREPARE votes into the network.
$SendPrepareVote(index) \triangleq$
    $SendMsg(\{[$
        $type \mapsto$ "PREPARE",
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round,$
        $index \mapsto index,$
        $cp\_round \mapsto 0,$
        $cp\_val \mapsto 0]\})$

SendPrecommitVote is used to broadcast PRECOMMIT votes into the network.
$SendPrecommitVote(index) \triangleq$
    $SendMsg(\{[$
        $type \mapsto$ "PRECOMMIT",
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round,$
        $index \mapsto index,$
        $cp\_round \mapsto 0,$
        $cp\_val \mapsto 0]\})$

SendCPPreVote is used to broadcast CP : PRE − VOTE votes into the network.
$SendCPPreVote(index,\ cp\_val) \triangleq$
    $SendMsg(\{[$
        $type \mapsto$ "CP:PRE-VOTE",
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round,$
        $index \mapsto index,$
        $cp\_round \mapsto states[index].cp\_round,$
        $cp\_val \mapsto cp\_val]\})$

SendCPMainVote is used to broadcast CP : MAIN − VOTE votes into the network.
$SendCPMainVote(index,\ cp\_val) \triangleq$
    $SendMsg(\{[$

```
        type      ↦ "CP:MAIN-VOTE",
        height    ↦ states[index].height,
        round     ↦ states[index].round,
        index     ↦ index,
        cp_round  ↦ states[index].cp_round,
        cp_val    ↦ cp_val]})

SendCPVotesForNextRound(index, cp_val) ≜
    SendMsg({
    [
        type      ↦ "CP:PRE-VOTE",
        height    ↦ states[index].height,
        round     ↦ states[index].round,
        index     ↦ index,
        cp_round  ↦ states[index].cp_round + 1,
        cp_val    ↦ cp_val],
    [
        type      ↦ "CP:MAIN-VOTE",
        height    ↦ states[index].height,
        round     ↦ states[index].round,
        index     ↦ index,
        cp_round  ↦ states[index].cp_round + 1,
        cp_val    ↦ cp_val]})

 AnnounceBlock is used to broadcast BLOCK − ANNOUNCE messages into the network.
AnnounceBlock(index) ≜
    SendMsg({[
        type      ↦ "BLOCK-ANNOUNCE",
        height    ↦ states[index].height,
        round     ↦ states[index].round,
        index     ↦ index,
        cp_round  ↦ 0,
        cp_val    ↦ 0]})
```

States functions

NewHeight state

```
NewHeight(index) ≜
    IF states[index].height ≥ MaxHeight
     THEN UNCHANGED ⟨states, log⟩
     ELSE
        ∧ ¬IsFaulty(index)
        ∧ states[index].name = "new-height"
        ∧ states[index].height < MaxHeight
        ∧ states′ = [states EXCEPT
```

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

 Propose state
Propose(index) ≜
    ∧  ¬IsFaulty(index)
    ∧  states[index].name = "propose"
    ∧  IF IsProposer(index)
       THEN SendProposal(index)
       ELSE UNCHANGED ⟨log⟩
    ∧  states' = [states EXCEPT
       ![index].name = "prepare",
       ![index].timeout = FALSE,
       ![index].cp_round = 0]

 Prepare state
Prepare(index) ≜
    ∧  ¬IsFaulty(index)
    ∧  states[index].name = "prepare"
    ∧  IF HasPrepareQuorum(index)
       THEN ∧ states' = [states EXCEPT ![index].name = "precommit"]
            ∧ UNCHANGED ⟨log⟩
       ELSE ∧ HasProposal(index)
            ∧ SendPrepareVote(index)
            ∧ UNCHANGED ⟨states⟩

 Precommit state
Precommit(index) ≜
    ∧ ¬IsFaulty(index)
    ∧ states[index].name = "precommit"
    ∧ IF HasPrecommitQuorum(index)
      THEN ∧ states' = [states EXCEPT ![index].name = "commit"]
           ∧ UNCHANGED ⟨log⟩
      ELSE ∧ HasProposal(index)
           ∧ SendPrecommitVote(index)
           ∧ UNCHANGED ⟨states⟩

 Commit state
Commit(index) ≜
    ∧ ¬IsFaulty(index)
    ∧ states[index].name = "commit"
    ∧ AnnounceBlock(index)
    ∧ states' = [states EXCEPT
```

```
        ![index].name = "new-height"]

 Timeout : A non − faulty Replica try to change the proposer if its timer expires.
Timeout(index) ≜
    ∧   ¬IsFaulty(index)
    ∧   states[index].round < MaxRound
    ∧   states[index].timeout = FALSE
    ∧
        ∨
            ∧ states[index].name = "prepare"
            ∧ SendCPPreVote(index, 1)

        ∨
            ∧ states[index].name = "precommit"
            ∧ SendCPPreVote(index, 0)
    ∧ states' = [states EXCEPT
            ![index].name = "cp:main-vote",
            ![index].timeout = TRUE]


CPPreVote(index) ≜
    ∧ ¬IsFaulty(index)
    ∧ states[index].name = "cp:pre-vote"
    ∧
        ∨
            ∧ CPHasOneMainVotesOneInPrvRound(index)
            ∧ SendCPPreVote(index, 1)
        ∨
            ∧ CPHasOneMainVotesZeroInPrvRound(index)
            ∧ SendCPPreVote(index, 0)
        ∨
            ∧ CPAllMainVotesAbstainInPrvRound(index)
            ∧ SendCPPreVote(index, 0)  biased to zero
    ∧ states' = [states EXCEPT ![index].name = "cp:main-vote"]

CPMainVote(index) ≜
    ∧ ¬IsFaulty(index)
    ∧ states[index].name = "cp:main-vote"
    ∧ CPHasPreVotesQuorum(index)
    ∧
        ∨
                all votes for 1
            ∧ CPHasPreVotesQuorumForOne(index)
            ∧ SendCPMainVote(index, 1)
            ∧ states' = [states EXCEPT ![index].name = "cp:decide"]
```

```
        ∨
              all votes for 0
            ∧ CPHasPreVotesQuorumForZero(index)
            ∧ SendCPMainVote(index, 0)
            ∧ states′ = [states EXCEPT ![index].name = "cp:decide"]
        ∨
              Abstain vote
            ∧ CPHasPreVotesForZeroAndOne(index)
            ∧ SendCPMainVote(index, 2)
            ∧ states′ = [states EXCEPT ![index].name = "cp:decide"]

CPDecide(index) ≜
    ∧ ¬IsFaulty(index)
    ∧ states[index].name = "cp:decide"
    ∧
        ∨
            ∧ states[index].cp_decided = 1
            ∧ states′ = [states EXCEPT ![index].name = "propose",
                                       ![index].round = states[index].round + 1]
        ∨
            ∧ states[index].cp_decided = 0
            ∧ states′ = [states EXCEPT ![index].name = "prepare"]
        ∨
            ∧ states[index].cp_decided = −1
            ∧ CPHasMainVotesQuorum(index)
            ∧
                IF   ∧ CPHasMainVotesQuorumForOne(index)
                     ∧ states[index].cp_round ≠ MaxCPRound − 1
                THEN states′ = [states EXCEPT ![index].name = "cp:pre-vote",
                                              ![index].cp_decided = 1,
                                              ![index].cp_round = states[index].cp_round + 1]
                ELSE IF ∨ CPHasMainVotesQuorumForZero(index)
                        ∨ states[index].cp_round = MaxCPRound − 1
                     THEN states′ = [states EXCEPT ![index].name = "cp:pre-vote",
                                                   ![index].cp_decided = 0,
                                                   ![index].cp_round = states[index].cp_round + 1]
                     ELSE states′ = [states EXCEPT ![index].name = "cp:pre-vote",
                                                   ![index].cp_round = states[index].cp_round + 1]

    ∧ log′ = log

Sync(index) ≜
    ∧ ¬IsFaulty(index)
    ∧
        ∨ states[index].name = "cp:pre-vote"
```

$\lor\ states[index].name = \text{“cp:main-vote”}$
$\lor\ states[index].name = \text{“cp:decide”}$
$\land\ HasBlockAnnounce(index)$
$\land\ states' = [states \text{ EXCEPT } ![index].name = \text{“prepare”}]$
$\land\ log' = log$

$Init \triangleq$
$\land\ log = \{\}$
$\land\ states = [index \in 0\,..\,Replicas - 1 \mapsto [$
$name \mapsto \text{“new-height”},$
$height \mapsto 0,$
$round \mapsto 0,$
$timeout \mapsto \text{FALSE},$
$cp\_round \mapsto 0,$
$cp\_decided \mapsto -1]]$

$Next \triangleq$
$\exists\, index \in 0\,..\,Replicas - 1 :$
$\lor\ NewHeight(index)$
$\lor\ Propose(index)$
$\lor\ Prepare(index)$
$\lor\ Precommit(index)$
$\lor\ Timeout(index)$
$\lor\ Commit(index)$
$\lor\ Sync(index)$
$\lor\ CPPreVote(index)$
$\lor\ CPMainVote(index)$
$\lor\ CPDecide(index)$

$Spec \triangleq$
$Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)$

Success : All non − faulty nodes eventually commit at MaxHeight.

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

TypeOK is the type − correctness invariant.

$TypeOK \triangleq$
$\land\ \ \forall\, index \in 0\,..\,Replicas - 1 :$
$\land\ states[index].name \in \{\text{“new-height”}, \text{“propose”}, \text{“prepare”},$
$\text{“precommit”}, \text{“commit”}, \text{“cp:pre-vote”}, \text{“cp:main-vote”}, \text{“cp:decide”}\}$
$\land\ states[index].height \leq MaxHeight$
$\land\ states[index].round \leq MaxRound$
$\land\ states[index].cp\_round \leq MaxCPRound + 1$

$\land\ states[index].name = \text{“new-height”} \land states[index].height > 1 \Rightarrow$
  $\land\ IsCommitted(states[index].height - 1)$
$\land\ states[index].name = \text{“precommit”} \Rightarrow$
  $\land\ HasPrepareQuorum(index)$
  $\land\ HasProposal(index)$
$\land\ states[index].name = \text{“commit”} \Rightarrow$
  $\land\ HasPrepareQuorum(index)$
  $\land\ HasPrecommitQuorum(index)$
  $\land\ HasProposal(index)$
$\land\ \forall\, round \in 0\,..\,states[index].round :$
  Not more than one proposal per round
  $\land\ Cardinality(GetProposal(states[index].height,\ round)) \leq 1$