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 limits the range of behaviors evaluated by *TLC*

MaxHeight,

The maximum number of round per height.
This limits the range of behaviors evaluated by *TLC*

MaxRound,

The maximum number of cp-round per height.
This limits the range of behaviors evaluated by *TLC*

MaxCPRound,

The total number of nodes in the network, denoted as *n in the protocol*.

NumNodes,

The total number of faulty nodes, *denoted as f in the protocol*.

f,

The total number of faulty nodes, *denoted as f in the protocol*.

t,

The indices of faulty nodes.

FaultyNodes

VARIABLES

log is a set of messages received by the system.

log,

states represents the state of each replica in the consensus protocol.

states

ThreeFPlusOne is equal to $3f + 1$, where f is the number of faulty nodes.

$ThreeFPlusOne \triangleq (3 * f) + 1$

TwoFPlusOne is equal to $2f + 1$, where f is the number of faulty nodes.

$TwoFPlusOne \triangleq (2 * f) + 1$

OneFPlusOne is equal to $f + 1$, where f is the number of faulty nodes.

$OneFPlusOne \triangleq (1 * f) + 1$

FourTPlusOne is equal to $3f + 1$, where f is the number of faulty nodes.

$FourTPlusOne \triangleq (4 * t) + 1$

TwoFPlusOne is equal to $2f + 1$, where f is the number of faulty nodes.

$ThreeTPlusOne \triangleq (3 * t) + 1$

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

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

ASSUME

Ensure that the number of nodes is sufficient to tolerate the specified number of faults.

```
    ∧ NumNodes ≥ ThreeFPlusOne
     Ensure that FaultyNodes is a valid subset of node indices.
    ∧ FaultyNodes ⊆ 0 . . NumNodes − 1
```

```
Helper functions

 Fetch a subset of messages in the network based on the params filter.
SubsetOfMsgs(params) ≜
   {msg ∈ log : ∀ field ∈ 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) ≜
    states[index].round%NumNodes = index

 Helper function to check if a node is faulty or not.
IsFaulty(index) ≜ index ∈ FaultyNodes

 HasPrepareAbsoluteQuorum checks whether the node with the given index
 has received all the PREPARE votes in this round.
HasPrepareAbsoluteQuorum(index) ≜
    Cardinality(SubsetOfMsgs([
        type     ↦ "PREPARE",
        height   ↦ states[index].height,
        round    ↦ states[index].round])) ≥ FourTPlusOne

 HasPrepareQuorum checks whether the node with the given index
 has received 2f + 1 the PREPARE votes in this round.
HasPrepareQuorum(index) ≜
    Cardinality(SubsetOfMsgs([
        type     ↦ "PREPARE",
        height   ↦ states[index].height,
        round    ↦ states[index].round])) ≥ ThreeTPlusOne

 HasPrecommitQuorum checks whether the node with the given index
 has received 2f + 1 the PRECOMMIT votes in this round.
HasPrecommitQuorum(index) ≜
    Cardinality(SubsetOfMsgs([
        type     ↦ "PRECOMMIT",
        height   ↦ states[index].height,
        round    ↦ states[index].round])) ≥ ThreeTPlusOne

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

CPHasPreVotesQuorum(index) ≜
    Cardinality(SubsetOfMsgs([
        type     ↦ "CP:PRE-VOTE",
        height   ↦ states[index].height,
        round    ↦ states[index].round,
        cp_round ↦ states[index].cp_round])) ≥ TwoFPlusOne

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

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

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

CPHasAMainVotesZeroInPrvRound(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

CPHasAMainVotesOneInPrvRound(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])) ≥ TwoFPlusOne

CPOneFPlusOneMainVotesAbstainInPrvRound(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])) ≥ OneFPlusOne

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

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])) ≥ TwoFPlusOne

CPHasMainVotesQuorumForZero(index) ≜
    Cardinality(SubsetOfMsgs([
        type     ↦ "CP:MAIN-VOTE",
        height   ↦ states[index].height,
        $round \mapsto states[index].round,$
        $cp\_round \mapsto states[index].cp\_round,$
        $cp\_val \mapsto 0])) \geq TwoFPlusOne$

$CPHasDecideVotesForZero(index) \triangleq$
    $Cardinality(SubsetOfMsgs([$
        $type \mapsto$ “CP:DECIDE”,
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round,$
        $cp\_val \mapsto 0])) > 0$

$CPHasDecideVotesForOne(index) \triangleq$
    $Cardinality(SubsetOfMsgs([$
        $type \mapsto$ “CP:DECIDE”,
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round,$
        $cp\_val \mapsto 1])) > 0$

$GetProposal(height, round) \triangleq$
    $SubsetOfMsgs([type \mapsto$ “PROPOSAL”, $height \mapsto height, round \mapsto round])$

$HasProposal(index) \triangleq$
    $Cardinality(GetProposal(states[index].height, states[index].round)) > 0$

$HasPrepared(index) \triangleq$
    $Cardinality(SubsetOfMsgs([$
        $type \mapsto$ “PREPARE”,
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round,$
        $index \mapsto index])) = 1$

$HasBlockAnnounce(index) \triangleq$
    $Cardinality(SubsetOfMsgs([$
        $type \mapsto$ “BLOCK-ANNOUNCE”,
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round])) \geq 1$

 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.
$IsCommitted \triangleq$
    LET $subset \triangleq SubsetOfMsgs([$
            $type \mapsto$ “BLOCK-ANNOUNCE”,
            $height \mapsto MaxHeight])$
    IN  $\land Cardinality(subset) \geq TwoFPlusOne$
        $\land \forall m1, m2 \in subset : m1.round = m2.round$
```

Network functions

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

```
$SendMsg(msg) \triangleq$
    IF $msg.cp\_round < MaxCPRound$
     THEN $log' = log \cup \{msg\}$
     ELSE $log' = log$
```

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.
$SendCPMainVote(index, cp\_val) \triangleq$
    $SendMsg([$
        $type \mapsto$ "CP:MAIN-VOTE",
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round,$
        $index \mapsto index,$
        $cp\_round \mapsto states[index].cp\_round,$
        $cp\_val \mapsto cp\_val])$

 SendCPDeciedVote is used to broadcast CP : DECIDE votes into the network.
$SendCPDeciedVote(index, cp\_val) \triangleq$
    $SendMsg([$
        $type \mapsto$ "CP:DECIDE",
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round,$
        $cp\_round \mapsto states[index].cp\_round,$
        $index \mapsto -1,$   reduce the model size
        $cp\_val \mapsto cp\_val])$

 AnnounceBlock is used to broadcast BLOCK − ANNOUNCE messages into the network.
$AnnounceBlock(index) \triangleq$
    $SendMsg([$
        $type \mapsto$ "BLOCK-ANNOUNCE",
        $height \mapsto states[index].height,$
        $round \mapsto states[index].round,$
        $index \mapsto index,$
        $cp\_round \mapsto 0,$
        $cp\_val \mapsto 0])$
```

States functions

```
 NewHeight state
$NewHeight(index) \triangleq$
    IF $states[index].height \geq MaxHeight$
     THEN UNCHANGED $\langle states, log \rangle$
     ELSE
        $\land \neg IsFaulty(index)$
        $\land states[index].name =$ "new-height"
        $\land states' = [states$ EXCEPT
            $![index].name =$ "propose",
            $![index].height = states[index].height + 1,$
            $![index].round = 0]$
        $\land\ log' = log$

Propose state
$Propose(index)\ \triangleq$
    $\land\ \neg IsFaulty(index)$
    $\land\ states[index].name =$ "propose"
    $\land$ IF $IsProposer(index)$
         THEN $SendProposal(index)$
         ELSE $log' = log$
    $\land\ states' = [states$ EXCEPT
         $![index].name =$ "prepare",
         $![index].cp\_round = 0]$

Prepare state
$Prepare(index)\ \triangleq$
    $\land\ \neg IsFaulty(index)$
    $\land\ states[index].name =$ "prepare"
    $\land\ HasProposal(index)$
    $\land\ SendPrepareVote(index)$
    $\land\ states' = states$

Precommit state
$Precommit(index)\ \triangleq$
    $\land \neg IsFaulty(index)$
    $\land\ states[index].name =$ "precommit"
    $\land$ IF $HasPrecommitQuorum(index)$
       THEN $\land\ states' = [states$ EXCEPT $![index].name =$ "commit"$]$
            $\land\ log' = log$
       ELSE $\land\ HasProposal(index)$
            $\land\ SendPrecommitVote(index)$
            $\land\ states' = states$

Commit state
$Commit(index)\ \triangleq$
    $\land \neg IsFaulty(index)$
    $\land\ states[index].name =$ "commit"
    $\land\ AnnounceBlock(index)$
    $\land\ states' = [states$ EXCEPT
       $![index].name =$ "new-height"$]$

Timeout : A non - faulty Replica try to change the proposer if its timer expires.
$Timeout(index)\ \triangleq$
    $\land\ \neg IsFaulty(index)$
    $\land\ states[index].name =$ "prepare"
    $\land\ states[index].round < MaxRound$
    ∧  $states' = [states$ EXCEPT $![index].name =$ "cp:pre-vote"$]$
    ∧  $log' = log$

$CPPreVote(index) \triangleq$
    ∧ $\neg IsFaulty(index)$
    ∧ $states[index].name =$ "cp:pre-vote"
       ∧ IF $states[index].cp\_round = 0$
           THEN
               IF $HasPrepareQuorum(index)$
                THEN ∧ $SendCPPreVote(index, 0)$
                     ∧ $states' = [states$ EXCEPT $![index].name =$ "cp:main-vote"$]$
                ELSE IF $HasPrepared(index)$
                       THEN ∧ $CPHasPreVotesMinorityQuorum(index)$
                            ∧ $SendCPPreVote(index, 1)$
                            ∧ $states' = [states$ EXCEPT $![index].name =$ "cp:main-vote"$]$
                       ELSE ∧ $SendCPPreVote(index, 1)$
                            ∧ $states' = [states$ EXCEPT $![index].name =$ "cp:main-vote"$]$
           ELSE
               ∧
                   ∨
                       ∧ $CPHasAMainVotesOneInPrvRound(index)$
                       ∧ $SendCPPreVote(index, 1)$
                   ∨
                       ∧ $CPHasAMainVotesZeroInPrvRound(index)$
                       ∧ $SendCPPreVote(index, 0)$
                   ∨
                       ∧ $CPAllMainVotesAbstainInPrvRound(index)$
                       ∧ $SendCPPreVote(index, 0)$  biased to zero
               ∧ $states' = [states$ EXCEPT $![index].name =$ "cp:main-vote"$]$

$CPMainVote(index) \triangleq$
    ∧ $\neg IsFaulty(index)$
    ∧ $states[index].name =$ "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
            ∧ $CPHasPreVotesQuorumForZero(index)$
            ∧ $SendCPMainVote(index, 0)$
        $\land$ $states' = [states$ EXCEPT $![index].name =$ "cp:decide"$]$
    $\lor$
          Abstain vote
        $\land$ $CPHasPreVotesForZeroAndOne(index)$
        $\land$ $SendCPMainVote(index, 2)$
        $\land$ $states' = [states$ EXCEPT $![index].name =$ "cp:decide"$]$

$CPDecide(index) \triangleq$
  $\land$ $\neg IsFaulty(index)$
  $\land$ $states[index].name =$ "cp:decide"
  $\land$ $CPHasMainVotesQuorum(index)$
  $\land$
    IF $CPHasMainVotesQuorumForZero(index)$
     THEN
        $\land$ $SendCPDeciedVote(index, 0)$
        $\land$ $states' = states$
     ELSE IF $CPHasMainVotesQuorumForOne(index)$
     THEN
        $\land$ $SendCPDeciedVote(index, 1)$
        $\land$ $states' = states$
     ELSE
        $\land$ $states' = [states$ EXCEPT $![index].name =$ "cp:pre-vote",
                                   $![index].cp\_round = states[index].cp\_round + 1]$
        $\land$ $log' = log$

$CPStrongTerminate(index) \triangleq$
  $\land$ $\neg IsFaulty(index)$
  $\land$
     $\lor$ $states[index].name =$ "cp:pre-vote"
     $\lor$ $states[index].name =$ "cp:main-vote"
     $\lor$ $states[index].name =$ "cp:decide"
  $\land$
    IF $CPHasDecideVotesForOne(index)$
     THEN $\land$ $states' = [states$ EXCEPT $![index].name =$ "propose",
                                    $![index].round = states[index].round + 1]$
          $\land$ $log' = log$
     ELSE IF $CPHasDecideVotesForZero(index)$
     THEN
         $\land$ $states' = [states$ EXCEPT $![index].name =$ "precommit"$]$
         $\land$ $log' = log$
     ELSE IF $\land$ $states[index].cp\_round = MaxCPRound$
             $\land$ $CPOneFPlusOneMainVotesAbstainInPrvRound(index)$
     THEN
        $\land$ $states' = [states$ EXCEPT $![index].name =$ "precommit"$]$
        $\land$ $log' = log$
        ELSE
            $\land$ $states' = states$
            $\land$ $log' = log$

$StrongCommit(index) \triangleq$
    $\land$ $\neg IsFaulty(index)$
    $\land$
        $\lor$ $states[index].name =$ "prepare"
        $\lor$ $states[index].name =$ "precommit"
        $\lor$ $states[index].name =$ "cp:pre-vote"
        $\lor$ $states[index].name =$ "cp:main-vote"
        $\lor$ $states[index].name =$ "cp:decide"
    $\land$ $HasPrepareAbsoluteQuorum(index)$
    $\land$ $states' = [states$ EXCEPT $![index].name =$ "commit"$]$
    $\land$ $log' = log$
```

```
$Init \triangleq$
    $\land$ $log = \{\}$
    $\land$ $states = [index \in 0 \,..\, NumNodes - 1 \mapsto [$
        $name$        $\mapsto$ "new-height",
        $height$      $\mapsto 0,$
        $round$       $\mapsto 0,$
        $cp\_round$   $\mapsto 0]]$

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

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

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

$Success \triangleq \Diamond(IsCommitted)$

TypeOK is the type − correctness invariant.

```
$TypeOK \triangleq$
    $\land \quad \forall\, index \in 0 \,..\, NumNodes - 1 :$
            $\land\ states[index].name \in \{$"new-height", "propose", "prepare",
              "precommit", "commit", "cp:pre-vote", "cp:main-vote", "cp:decide"$\}$
            $\land\ states[index].height \leq MaxHeight$
            $\land\ states[index].round \leq MaxRound$
            $\land\ states[index].cp\_round \leq MaxCPRound$
            $\land\ states[index].name =$ "new-height" $\land\ states[index].height > 0 \Rightarrow$
                $\land\ HasBlockAnnounce(index)$
            $\land\ states[index].name =$ "precommit" $\Rightarrow$
                $\land\ HasPrepareQuorum(index)$
                $\land\ HasProposal(index)$
            $\land\ states[index].name =$ "commit" $\Rightarrow$
                $\land\ HasPrepareQuorum(index)$
                $\land\ HasProposal(index)$
            $\land\ \forall\, round \in 0 \,..\, states[index].round :$
                *Not more than one proposal per round*
                $\land\ Cardinality(GetProposal(states[index].height,\ round)) \leq 1$
```