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.*
$\land FaultyNodes \subseteq 0 \,..\, N - 1$
*Ensure that the number of faulty nodes does not exceed the maximum allowed.*
$\land Cardinality(FaultyNodes) \leq F$
*Ensure that MaxRound is greater than 0.*
$\land MaxRound > 0$

*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.*

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

*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*

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

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

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

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

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

*Check if the node has received at least $2f + 1$ $CP : PRE - VOTE$ votes in the current CP round.*

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

*Check if the node has received at least $2f + 1$ $CP : PRE - VOTE$ votes with value $1(yes)$in the current CP round.*

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

*Check if the node has received at least $2f + 1$ $CP : PRE - VOTE$ votes with value $0(no)$in the current CP round.*

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

*Check if the node has received at least $f + 1$ $CP : PRE - VOTE$ votes with value $1(yes)$in the current CP round.*

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

*Check if the node has received both yes and no $CP:PRE-VOTE$ votes in the current CP round.*
$CPHasPreVotesForYesAndNo(index) \triangleq$
$\wedge Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:PRE-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round,$
$cp\_val \mapsto 0])) \geq 1$
$\wedge Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:PRE-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round,$
$cp\_val \mapsto 1])) \geq 1$

*Check if the node has received at least one $CP:MAIN-VOTE$ with value $0(no)$in the previous CP round.*
$CPHasOneMainVotesNoInPrvRound(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:MAIN-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round - 1,$
$cp\_val \mapsto 0])) > 0$

*Check if the node has received at least one $CP:MAIN-VOTE$ with value $1(yes)$in the previous CP round.*
$CPHasOneMainVotesYesInPrvRound(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:MAIN-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round - 1,$
$cp\_val \mapsto 1])) > 0$

*Check if the node has received at least $2f + 1$ $CP:MAIN-VOTE$ votes with value $2(abstain)$in the previous CP round.*
$CPAllMainVotesAbstainInPrvRound(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$type \mapsto$ "CP:MAIN-VOTE",
$round \mapsto states[index].round,$
$cp\_round \mapsto states[index].cp\_round - 1,$
$cp\_val \mapsto 2])) \geq TwoFPlusOne$

*Check if the node has received at least $2f + 1$ $CP:MAIN-VOTE$ votes in the current CP round.*
$CPHasMainVotesQuorum(index) \triangleq$

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

*Check if the node has received at least* $2f + 1$ $CP : MAIN - VOTE$ *votes with value* $1(yes)$*in the current CP round.*
$CPHasMainVotesQuorumForYes(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "CP:MAIN-VOTE",
$\quad round \mapsto states[index].round,$
$\quad cp\_round \mapsto states[index].cp\_round,$
$\quad cp\_val \mapsto 1])) \geq TwoFPlusOne$

*Check if the node has received at least* $2f + 1$ $CP : MAIN - VOTE$ *votes with value* $2(abstain)$*in the current CP round.*
$CPHasMainVotesQuorumForAbstain(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "CP:MAIN-VOTE",
$\quad round \mapsto states[index].round,$
$\quad cp\_round \mapsto states[index].cp\_round,$
$\quad cp\_val \mapsto 2])) \geq TwoFPlusOne$

*Check if the node has received at least one* $CP : DECIDED$ *vote with value* $1(yes)$*in the current round.*
$CPHasDecideVotesForYes(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "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",
$\quad round \mapsto states[index].round])) > 0$

*Check if the node has sent its own PRECOMMIT vote in the current round.*
$HasPrecommited(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "PRECOMMIT",
$\quad round \mapsto states[index].round,$
$\quad index \mapsto index])) = 1$

*Check if the node has received an announcement message in the current round.*
$HasAnnouncement(index) \triangleq$
$Cardinality(SubsetOfMsgs(logs[index], [$
$\quad type \mapsto$ "ANNOUNCEMENT",
$\quad round \mapsto states[index].round])) > 0$

*Check if the proposal is committed.*
*A proposal is considered committed if a super − majority of non − faulty replicas announce the same proposal.*

$IsCommitted \triangleq$
LET $subset \triangleq SubsetOfMsgs(network, [type \mapsto$ "ANNOUNCEMENT"$])$
IN $\land Cardinality(subset) \geq TwoFPlusOne$
$\land \forall\, m1,\, m2 \in subset : m1.round = m2.round$

---

*Network functions*

*Simulate a replica sending a message by appending it to the network*
*The message is delivered to the sender* s log immediately.

$SendMsg(msg) \triangleq$
$\land \quad network' = network \cup \{msg\}$
$\land \quad logs' = [logs$ EXCEPT $![msg.index] = logs[msg.index] \cup \{msg\}]$

Deliver a message to the specified replica's log.

$DeliverMsg(index) \triangleq$
LET $undeliveredMsgs \triangleq network \setminus logs[index]$
IN IF $Cardinality(undeliveredMsgs) = 0$ THEN
UNCHANGED $\langle vars \rangle$
ELSE
LET $msg \triangleq$ CHOOSE $x \in undeliveredMsgs :$ TRUE
IN
$\land\, logs' = [logs$ EXCEPT $![index] = logs[index] \cup \{msg\}]$
$\land$ UNCHANGED $\langle states,\, network \rangle$

Broadcast a *PROPOSAL* message into the network.

$SendProposal(index) \triangleq$
$SendMsg([$
$type \quad \mapsto$ "PROPOSAL",
$round \quad \mapsto states[index].round,$
$index \quad \mapsto index,$
$cp\_round \mapsto 0,$
$cp\_val \quad \mapsto 0])$

Broadcast *PRECOMMIT* votes into the network.

$SendPreCommitVote(index) \triangleq$
$SendMsg([$
$type \quad \mapsto$ "PRECOMMIT",
$round \quad \mapsto states[index].round,$
$index \quad \mapsto index,$
$cp\_round \mapsto 0,$
$cp\_val \quad \mapsto 0])$

Broadcast *CP*:PRE-VOTE votes into the network.

$SendCPPreVote(index,\, cp\_val) \triangleq$

```
    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      ↦ index,
        cp_val     ↦ cp_val])

 Broadcast ANNOUNCEMENT messages into the network.
Announce(index) ≜
    SendMsg([
        type       ↦ "ANNOUNCEMENT",
        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⟩
```

$\land\ states' = [states \text{ EXCEPT } ![index].name = \text{“precommit”}]$

Transition to the *precommit* state.

$PreCommit(index) \triangleq$

$\land\ \neg IsFaulty(index)$
$\land\ states[index].name = \text{“precommit”}$
$\land\ HasProposal(index)$
$\land\ SendPreCommitVote(index)$
$\land\ states' = states$

*AbsoluteCommit* checks if $3F + 1$ replicas voted for the proposal.

$AbsoluteCommit(index) \triangleq$

$\land\ \neg IsFaulty(index)$
$\land\ states[index].name \neq \text{“commit”}$ to prevent shuttering
$\land\ HasPreCommitAbsolute(index)$
$\land\ states' = [states \text{ EXCEPT } ![index].name = \text{“commit”}]$
$\land\ \text{UNCHANGED } \langle network, logs \rangle$

*QuorumCommit* checks if $2F + 1$ replicas voted for the proposal after the change-proposer phase.

$QuorumCommit(index) \triangleq$

$\land\ \neg IsFaulty(index)$
$\land\ states[index].name = \text{“precommit”}$
$\land\ states[index].decided = \text{TRUE}$
$\land\ HasPreCommitQuorum(index)$
$\land\ states' = [states \text{ EXCEPT } ![index].name = \text{“commit”}]$
$\land\ \text{UNCHANGED } \langle network, logs \rangle$

Transition to the commit state.

$Commit(index) \triangleq$

$\land\ \neg IsFaulty(index)$
$\land\ states[index].name = \text{“commit”}$
$\land\ Announce(index)$
$\land\ \text{UNCHANGED } \langle states \rangle$

Transition for timeout: a non-faulty replica changes the proposer if its timer expires.

$Timeout(index) \triangleq$

$\land\ \ \neg IsFaulty(index)$
$\land\ \ states[index].name = \text{“precommit”}$
$\land\ \ states[index].decided = \text{FALSE}$
$\land$

To limit the the behaviors.

$\lor\ states[index].round < MaxRound$
$\lor\ HasPreCommitQuorum(index)$

$\land\ states' = [states \text{ EXCEPT } ![index].name = \text{“cp:pre-vote”}]$
$\land\ \text{UNCHANGED } \langle network, logs \rangle$

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 ¬HasPrecommited(index) THEN
                ∧ SendCPPreVote(index, 1)
                ∧ states' = [states EXCEPT ![index].name = "cp:main-vote"]
             ELSE IF Cardinality(
                                SubsetOfMsgs(logs[index], [type ↦ "PRECOMMIT",    round ↦ states[index].rou
                                SubsetOfMsgs(logs[index], [type ↦ "CP:PRE-VOTE", round  ↦ states[index].rou
                            ) ≥ TwoFPlusOne THEN
                        Check if there is quorum of PRECOMMIT votes
                        IF HasPreCommitQuorum(index) THEN
                            ∧ SendCPPreVote(index, 0)
                            ∧ states' = [states EXCEPT ![index].name = "cp:main-vote"]
                         ELSE
                            ∧ 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 when all votes abstain
            ∧ 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)
            ∧ states' = [states EXCEPT ![index].name = "precommit",
                                       ![index].decided = TRUE]
```

```
            ∧ 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)
    ∧
        ∨
            ∧ CPHasMainVotesQuorumForYes(index)
            ∧ SendCPDecideVote(index, 1)
            ∧ states′ = [states EXCEPT ![index].name = "propose",
                                       ![index].round = states[index].round + 1]
        ∨
            ∧ states[index].cp_round < MaxCPRound
            ∧ CPHasMainVotesQuorumForAbstain(index)
            ∧ 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"
    ∧
        To limit the the behaviors.
        IF  ∧ states[index].cp_round = MaxCPRound
            ∧ HasPreCommitQuorum(index) THEN
                ∧ states′ = [states EXCEPT ![index].name = "precommit",
                                           ![index].decided = TRUE]

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

$!\,[index].round = states[index].round + 1,$
$!\,[index].cp\_round = 0]$

ELSE
$\land\ states' = states$

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

---

Initial state

$Init \triangleq$
$\land\ network = \{\}$
$\land\ logs = [index \in 0\,..\,N-1 \mapsto \{\}]$
$\land\ states = [index \in 0\,..\,N-1 \mapsto [$
$name \mapsto$ "propose",
$decided \mapsto$ FALSE,
$round \mapsto 0,$
$cp\_round \mapsto 0]]$

State transition relation

$Next \triangleq$
$\exists\, index \in 0\,..\,N-1 :$
$\lor\ Propose(index)$
$\lor\ PreCommit(index)$
$\lor\ Timeout(index)$
$\lor\ Commit(index)$
$\lor\ AbsoluteCommit(index)$
$\lor\ QuorumCommit(index)$
$\lor\ CPPreVote(index)$
$\lor\ CPMainVote(index)$
$\lor\ CPDecide(index)$
$\lor\ CPStrongTerminate(index)$
$\lor\ DeliverMsg(index)$

Specification

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

Success: All non-faulty nodes eventually commit.

$Success \triangleq \Diamond(IsCommitted)$

*TypeOK* is the type-correctness invariant.

$TypeOK \triangleq$
$\land\quad \forall\, index \in 0\,..\,N-1 :$
$\land\ states[index].round \leq MaxRound$
$\land\ states[index].cp\_round \leq MaxCPRound$

$\land\ states[index].round \geq 0$
$\land\ states[index].cp\_round \geq 0$
$\land\ states[index].name \in \{$"propose", "precommit", "commit", "cp:pre-vote", "cp:main-vote", "cp:decid
$\land\ states[index].decided \in \{\text{TRUE}, \text{FALSE}\}$
$\land\ states[index].name =$ "propose" $\land\ states[index].round > 0 \Rightarrow$
  $\land\ Cardinality(SubsetOfMsgs(network, [$
    $type \quad \mapsto$ "CP:DECIDED",
    $round \quad \mapsto states[index].round - 1,$
    $cp\_val \quad \mapsto 1])) > 0$
  $\land\ Cardinality(SubsetOfMsgs(network, [$
    $type \quad \mapsto$ "ANNOUNCEMENT",
    $round \quad \mapsto states[index].round - 1])) = 0$
$\land\ states[index].name =$ "commit" $\Rightarrow$
  $\land\ Cardinality(SubsetOfMsgs(network, [$
    $type \quad \mapsto$ "PRECOMMIT",
    $round \quad \mapsto states[index].round])) \geq TwoFPlusOne$
  $\land\ Cardinality(SubsetOfMsgs(network, [$
    $type \quad \mapsto$ "PROPOSAL",
    $round \quad \mapsto states[index].round])) = 1$
  $\land\ \text{LET}\ subset \triangleq SubsetOfMsgs(network, [type \mapsto$ "ANNOUNCEMENT"$])$
    IN $\quad \land \forall\, m1, m2 \in subset : m1.round = m2.round$
$\land\quad \forall\, msg \in network :$
  $\land\ msg.round \leq MaxRound$
  $\land\ msg.cp\_round \leq MaxCPRound$
  $\land\ msg.round \geq 0$
  $\land\ msg.cp\_round \geq 0$
  $\land\ msg.type \in \{$"PROPOSAL", "PRECOMMIT", "CP:PRE-VOTE", "CP:MAIN-VOTE", "CP:DECIDED
  $\land\ msg.index \in 0\,..\,N-1$