```
──────────────── MODULE Proposal ────────────────
INSTANCE Naturals

INSTANCE FiniteSets

INSTANCE Sequences

INSTANCE TLC

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

  An empty constant
CONSTANT Nil

  Transaction type constants
CONSTANTS
   Change,
   Rollback

  Phase constants
CONSTANTS
   Initialize,
   Validate,
   Commit,
   Apply

Phase ≜
   {Initialize,
    Validate,
    Commit,
    Apply}

  Status constants
CONSTANTS
   InProgress,
   Complete,
   Failed

State ≜
   {InProgress,
    Complete,
    Failed}

CONSTANTS
   Valid,
   Invalid

CONSTANTS
```

```
    Success,
    Failure

  The set of all nodes
CONSTANT Node
```

```
  A record of per-target proposals
VARIABLE proposal

  A record of per-target configurations
VARIABLE configuration

  A record of target states
VARIABLE target

  A record of target masterships
VARIABLE mastership

Test ≜ INSTANCE Test WITH
    File         ← "Proposal.log",
    CurrState ← [
        proposals       ↦ proposal,
        configuration ↦ configuration,
        mastership     ↦ mastership,
        target            ↦ target],
    SuccState ← [
        proposals       ↦ proposal′,
        configuration ↦ configuration′,
        mastership     ↦ mastership′,
        target            ↦ target′]
```

```
  Reconcile a proposal
ReconcileProposal(n, i) ≜
    ∧ i ∈ DOMAIN proposal
    ∧ ∨ ∧ proposal[i].phase = Initialize
          ∧ proposal[i].state  = InProgress
          ∧ proposal′ = [proposal EXCEPT ![i].state = Complete]
          ∧ configuration′ = [configuration EXCEPT !.proposal.index = i]
          ∧ UNCHANGED ⟨target⟩
        While in the Validate phase, validate the proposed changes.
        If validation is successful, the proposal also records the changes
        required to roll back the proposal and the index to which to roll back.
       ∨ ∧ proposal[i].phase = Validate
          ∧ proposal[i].state  = InProgress
```

```
$\land$ $configuration.commit.index = i - 1$
    For Change proposals validate the set of requested changes.
$\land$ $\lor$ $\land$ $proposal[i].type = Change$
        $\land$ LET $rollbackIndex \triangleq configuration.config.index$
                $rollbackValues \triangleq [p \in$ DOMAIN $proposal[i].change.values \mapsto$
                                        IF $p \in$ DOMAIN $configuration.config.values$ THEN
                                            $configuration.config.values[p]$
                                         ELSE
                                            $[value \mapsto Nil,$
                                             $delete \mapsto$ TRUE$]]$
            Model validation successes and failures with Valid and Invalid results.
          IN  $\exists r \in \{Valid, Invalid\}$ :
                 If the Change is Valid, record the changes required to roll
                 back the proposal and the index to which the rollback changes
                 will roll back the configuration.
                $\lor$ $\land$ $r = Valid$
                   $\land$ $proposal' = [proposal$ EXCEPT $![i].rollback.index = rollbackIndex,$
                                                      $![i].rollback.values = rollbackValues,$
                                                      $![i].state = Complete]$
                $\lor$ $\land$ $r = Invalid$
                   $\land$ $proposal' = [proposal$ EXCEPT $![i].state = Failed]$
    For Rollback proposals, validate the rollback changes which are
    proposal being rolled back.
    $\lor$ $\land$ $proposal[i].type = Rollback$
            Rollbacks can only be performed on Change type proposals.
        $\land$ $\lor$ $\land$ $proposal[proposal[i].rollback.index].type = Change$
                  Only roll back the change if it's the lastest change made
                  to the configuration based on the configuration index.
              $\land$ $\lor$ $\land$ $configuration.config.index = proposal[i].rollback.index$
                    $\land$ LET $changeIndex \triangleq proposal[proposal[i].rollback.index].rollback.index$
                            $changeValues \triangleq proposal[proposal[i].rollback.index].rollback.values$
                            $rollbackValues \triangleq proposal[proposal[i].rollback.index].change.values$
                      IN  $\exists r \in \{Valid, Invalid\}$ :
                             If the Rollback is Valid, record the changes required to
                             roll back the target proposal and the index to which the
                             configuration is being rolled back.
                            $\lor$ $\land$ $r = Valid$
                               $\land$ $proposal' = [proposal$ EXCEPT $![i].change.index = changeIndex,$
                                                                  $![i].change.values = changeValues,$
                                                                  $![i].rollback.values = rollbackValues,$
                                                                  $![i].state = Complete]$
                            $\lor$ $\land$ $r = Invalid$
                               $\land$ $proposal' = [proposal$ EXCEPT $![i].state = Failed]$
                  If the Rollback target is not the most recent change to the configuration,
                  fail validation for the proposal.
                  $\lor \land configuration.config.index \neq proposal[i].rollback.index$
                     $\land proposal' = [proposal \text{ EXCEPT } ![i].state = Failed]$
                $\land \text{UNCHANGED } \langle configuration \rangle$
              If a Rollback proposal is attempting to roll back another Rollback,
              fail validation for the proposal.
              $\lor \land proposal[proposal[i].rollback.index].type = Rollback$
                 $\land proposal' = [proposal \text{ EXCEPT } ![i].state = Failed]$
   $\land \text{UNCHANGED } \langle configuration, target \rangle$
 If the proposal failed, set the configuration's commit index to the proposal index.
$\lor \land proposal[i].phase = Validate$
   $\land proposal[i].state = Failed$
   $\land configuration.commit.index = i - 1$
   $\land configuration' = [configuration \text{ EXCEPT } !.commit.index = i]$
   $\land \text{UNCHANGED } \langle proposal, target \rangle$
 While in the Commit state, commit the proposed changes to the configuration.
$\lor \land proposal[i].phase = Commit$
   $\land proposal[i].state = InProgress$
    Only commit the proposal if the prior proposal has already been committed.
   $\land configuration.commit.index = i - 1$
   $\land configuration' = [configuration \text{ EXCEPT } !.config.values = proposal[i].change.values,$
                                            $!.config.index = proposal[i].change.index,$
                                            $!.commit.index = i]$
   $\land proposal' = [proposal \text{ EXCEPT } ![i].state = Complete]$
   $\land \text{UNCHANGED } \langle target \rangle$
 While in the Apply phase, apply the proposed changes to the target.
$\lor \land proposal[i].phase = Apply$
   $\land proposal[i].state = InProgress$
   $\land configuration.target.index = i - 1$
   $\land configuration.target.term = mastership.term$
   $\land mastership.master = n$
    Model successful and failed target update requests.
   $\land \exists r \in \{Success, Failure\} :$
        $\lor \land r = Success$
           $\land target' = proposal[i].change.values \text{ @@ } target$
           $\land configuration' = [configuration \text{ EXCEPT}$
                                  $!.target.index = i,$
                                  $!.target.values = proposal[i].change.values$
                                     $\text{@@ } configuration.target.values]$
           $\land proposal' = [proposal \text{ EXCEPT } ![i].state = Complete]$
         If the proposal could not be applied, update the configuration's applied index
         and mark the proposal Failed.
        $\lor \land r = Failure$
           $\land configuration' = [configuration \text{ EXCEPT } !.target.index = i]$
           $\land proposal' = [proposal \text{ EXCEPT } ![i].state = Failed]$
           $\land \text{UNCHANGED } \langle target \rangle$
    ∧ UNCHANGED ⟨mastership⟩
```