```
┌──────────────── MODULE Proposal ────────────────┐
EXTENDS Configuration, Mastership

INSTANCE Naturals

INSTANCE FiniteSets

LOCAL INSTANCE TLC

├─────────────────────────────────────────────────┤

  Transaction type constants
CONSTANTS
   ProposalChange,
   ProposalRollback

  Phase constants
CONSTANTS
   ProposalCommit,
   ProposalApply

  Status constants
CONSTANTS
   ProposalInProgress,
   ProposalComplete,
   ProposalFailed

CONSTANT TraceProposal

  A record of per-target proposals
VARIABLE proposal

├─────────────────────────────────────────────────┤

LOCAL InitState ≜ [
   proposals      ↦ proposal,
   configurations ↦ configuration,
   targets         ↦ target,
   masterships     ↦ mastership,
   nodes          ↦ node]

LOCAL NextState ≜ [
   proposals      ↦ proposal′,
   configurations ↦ configuration′,
   targets         ↦ target′,
   masterships     ↦ mastership′,
   nodes          ↦ node′]
```

```
LOCAL Trace ≜ INSTANCE Trace WITH
   Module     ← "Proposal",
   InitState  ← InitState,
   NextState  ← NextState,
   Enabled    ← TraceProposal
```

```
  Reconcile a proposal
ReconcileProposal(n, i) ≜
     Only the master can process proposals for the target.
   ∧ mastership.master = n
       While in the Commit state, commit the proposed changes to the configuration.
   ∧ ∨ ∧ proposal[i].phase = ProposalCommit
        ∧ ∨ ∧ proposal[i].state = ProposalInProgress
               Only commit the proposal if the prior proposal has already been committed.
             ∧ configuration.committed.index = i − 1
                 For Change proposals validate the set of requested changes.
             ∧ ∨ ∧ proposal[i].type = ProposalChange
                        If all the change values are valid, record the changes required to roll
                        back the proposal and the index to which the rollback changes
                        will roll back the configuration.
                  ∧ ∨ LET rollbackIndex  ≜ configuration.committed.revision
                          rollbackValues ≜ [p ∈ DOMAIN proposal[i].change.values ↦
                                               IF p ∈ DOMAIN configuration.committed.values THEN
                                                  configuration.committed.values[p]
                                                ELSE
                                                  [index ↦ 0, value ↦ Nil]]
                          changeValues   ≜ [p ∈ DOMAIN proposal[i].change.values ↦
                                               proposal[i].change.values[p] @@ [index ↦ i]]
                      IN  ∧ configuration′ = [configuration EXCEPT !.committed.index    = i,
                                                                   !.committed.revision = i,
                                                                   !.committed.values   = changeValues]
                          ∧ proposal′ = [proposal EXCEPT ![i].change = [
                                                            index  ↦ i,
                                                            values ↦ changeValues],
                                                         ![i].rollback = [
                                                            index  ↦ rollbackIndex,
                                                            values ↦ rollbackValues],
                                                         ![i].state = ProposalComplete]
                      A proposal can fail validation at this point, in which case the proposal
                      is marked failed.
                    ∨ ∧ proposal′ = [proposal EXCEPT ![i].state = ProposalFailed]
                      ∧ UNCHANGED ⟨configuration⟩
                 For Rollback proposals, validate the rollback changes which are
```

```
            proposal being rolled back.
            $\lor \land proposal[i].type = ProposalRollback$
                Rollbacks can only be performed on Change type proposals.
              $\land \lor \land proposal[proposal[i].rollback.index].type = ProposalChange$
                      Only roll back the change if it's the latest change made
                      to the configuration based on the configuration index.
                    $\land \lor \land configuration.committed.revision = proposal[i].rollback.index$
                        Record the changes required to roll back the target proposal and the index to
                        which the configuration is being rolled back.
                        $\land$ LET $changeIndex \triangleq proposal[proposal[i].rollback.index].rollback.index$
                                $changeValues \triangleq proposal[proposal[i].rollback.index].rollback.values$
                              Note: these two changes must be implemented as an atomic, idempotent update.
                              Implementations should check if the configuration has already been updated and
                                skip the configuration update if the committed index is $\geq$ the proposal index.
                          IN  $\land configuration' = [configuration$ EXCEPT $!.committed.index \quad = i,$
                                                                       $!.committed.revision = changeI$
                                                                       $!.committed.values \quad = change$
                              $\land proposal' = [proposal$ EXCEPT $![i].change = [$
                                                                  $index \mapsto changeIndex,$
                                                                  $values \mapsto changeValues],$
                                                               $![i].state \ = ProposalComplete]$
                      If the Rollback target is not the most recent change to the configuration,
                      fail validation for the proposal.
                      $\lor \land configuration.committed.revision \neq proposal[i].rollback.index$
                          Note: these two changes must be implemented as an atomic, idempotent update.
                          Implementations should check if the configuration has already been updated and
                          skip the configuration update if the committed index is $\geq$ the proposal index.
                        $\land configuration' = [configuration$ EXCEPT $!.committed.index = i]$
                        $\land proposal' = [proposal$ EXCEPT $![i].state = ProposalFailed]$
                If a Rollback proposal is attempting to roll back another Rollback,
                fail validation for the proposal.
                $\lor \land proposal[proposal[i].rollback.index].type = ProposalRollback$
                    Note: these two changes must be implemented as an atomic, idempotent update.
                    Implementations should check if the configuration has already been updated and
                    skip the configuration update if the committed index is $\geq$ the proposal index.
                  $\land configuration' = [configuration$ EXCEPT $!.committed.index = i]$
                  $\land proposal' = [proposal$ EXCEPT $![i].state = ProposalFailed]$
          $\land$ UNCHANGED $\langle target \rangle$
        Once the proposal is committed, update the configuration's commit index
        and move to the apply phase.
        $\lor \land proposal[i].state = ProposalComplete$
          $\land proposal' = [proposal$ EXCEPT $![i].phase = ProposalApply,$
                                           $![i].state \ = ProposalInProgress]$
          $\land$ UNCHANGED $\langle configuration, target \rangle$
    While in the Apply phase, apply the proposed changes to the target.
```

```
      $\lor \land proposal[i].phase = ProposalApply$
         $\land configuration.applied.index = i - 1$
         $\land proposal[i].state = ProposalInProgress$
          Process the proposal once the prior proposal has been applied.
         $\land i - 1 \in \textsc{domain}\ proposal \Rightarrow$
                 $\lor \land proposal[i-1].phase = ProposalCommit$
                    $\land proposal[i-1].state\ \ = ProposalFailed$
                 $\lor \land proposal[i-1].phase = ProposalApply$
                    $\land proposal[i-1].state\ \ \in \{ProposalComplete, ProposalFailed\}$
          Verify the applied term is the current mastership term to ensure the
          configuration has been synchronized following restarts.
         $\land configuration.applied.term = mastership.term$
          Verify the node's connection to the target.
         $\land node[n].connected$
         $\land target.running$
          Model successful and failed target update requests.
         $\land \lor \land target' = [target\ \textsc{except}\ !.values = proposal[i].change.values]$
                Note: these two changes must be implemented as an atomic, idempotent update.
                Implementations should check if the configuration has already been updated and
                skip the configuration update if the applied index is $\geq$ the proposal index.
               $\land \textsc{let}\ index\ \ \triangleq\ proposal[i].change.index$
                       $values\ \triangleq\ proposal[i].change.values\ @@\ configuration.applied.values$
                  $\textsc{in}\ \ \ configuration' = [configuration\ \textsc{except}\ !.applied.index\ \ \ \ = i,$
                                                                    $!.applied.revision = index,$
                                                                    $!.applied.values\ \ \ = values]$
               $\land proposal' = [proposal\ \textsc{except}\ ![i].state = ProposalComplete]$
             If the proposal could not be applied, update the configuration's applied index
             and mark the proposal Failed.
             Note: these two changes must be implemented as an atomic, idempotent update.
             Implementations should check if the configuration has already been updated and
             skip the configuration update if the applied index is $\geq$ the proposal index.
            $\lor \land configuration' = [configuration\ \textsc{except}\ !.applied.index = i]$
               $\land proposal' = [proposal\ \textsc{except}\ ![i].state = ProposalFailed]$
               $\land \textsc{unchanged}\ \langle target \rangle$
   $\land \textsc{unchanged}\ \langle mastership, node \rangle$
```

Formal specification, constraints, and theorems.

```
$InitProposal \triangleq$
   $\land proposal = [$
       $i \in \{\} \mapsto [$
         $type\ \ \ \ \ \mapsto ProposalChange,$
         $change\ \ \mapsto [$
             $index\ \ \mapsto 0,$
```

```
          $values \mapsto [p \in \{\} \mapsto [index \mapsto 0, value \mapsto Nil, delete \mapsto \text{FALSE}]]],$
        $rollback \mapsto [$
          $index \ \ \mapsto 0,$
          $values \mapsto [p \in \{\} \mapsto [index \mapsto 0, value \mapsto Nil, delete \mapsto \text{FALSE}]]],$
        $phase \quad\ \mapsto ProposalCommit,$
        $state \quad\ \ \mapsto ProposalInProgress]]$
    $\land Trace!Init$

$NextProposal \triangleq$
    $\lor \exists n \in Nodes :$
        $\exists i \in \text{DOMAIN } proposal :$
          $Trace!Step(ReconcileProposal(n, i), [node \mapsto n, index \mapsto i])$
```

* Modification History

* Last modified *Fri Apr* 21 19:15:11 *PDT* 2023 by *jhalterm*

* Last modified *Mon Feb* 21 01:24:12 *PST* 2022 by *jordanhalterman*

* Created Sun *Feb* 20 10:07:16 *PST* 2022 by *jordanhalterman*