```
———————————————— MODULE Proposals ————————————————
EXTENDS Configurations, Mastership

INSTANCE Naturals

INSTANCE FiniteSets

LOCAL INSTANCE TLC

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

  Transaction type constants
CONSTANTS
   ProposalChange,
   ProposalRollback

  Phase constants
CONSTANTS
   ProposalInitialize,
   ProposalValidate,
   ProposalAbort,
   ProposalCommit,
   ProposalApply

  Status constants
CONSTANTS
   ProposalPending,
   ProposalInProgress,
   ProposalComplete,
   ProposalFailed

  A record of per-target proposals
VARIABLE proposal

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

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

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

```
LOCAL Trace ≜ INSTANCE Trace WITH
    Module    ← "Proposals",
    InitState ← InitState,
    NextState ← NextState
```

```
 Reconcile a proposal
ReconcileProposal(n, i) ≜
    ∧ ∨ ∧ proposal[i].phase = ProposalInitialize
          ∧ ∨ ∧ proposal[i].state = ProposalInProgress
                ∧ proposal′ = [proposal EXCEPT ![i].state = ProposalComplete]
                ∧ configuration′ = [configuration EXCEPT !.proposed.index = i]
                ∧ UNCHANGED ⟨target⟩
            ∨ ∧ proposal[i].state = ProposalComplete
                ∧ proposal′ = [proposal EXCEPT ![i].phase = ProposalValidate,
                                               ![i].state  = ProposalInProgress]
                ∧ UNCHANGED ⟨configuration, 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 = ProposalValidate
          ∧ ∨ ∧ proposal[i].state = ProposalInProgress
                ∧ configuration.index = i − 1
                   For Change proposals validate the set of requested changes.
                ∧ ∨ ∧ proposal[i].type = ProposalChange
                      ∧ LET rollbackIndex  ≜ configuration.committed.index
                            rollbackValues ≜ [p ∈ DOMAIN proposal[i].change.values ↦
                                               IF p ∈ DOMAIN configuration.committed.values THEN
                                                  configuration.committed.values[p]
                                                ELSE
                                                  [delete ↦ TRUE]]
                        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.
                        IN
                          ∨ proposal′ = [proposal EXCEPT ![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]
                   For Rollback proposals, validate the rollback changes which are
                   proposal being rolled back.
                  ∨ ∧ proposal[i].type = ProposalRollback
                Rollbacks can only be performed on Change type proposals.
             ∧ ∨ ∧ proposal[proposal[i].rollback.index].type = ProposalChange
                    Only roll back the change if it's the lastest change made
                    to the configuration based on the configuration index.
                 ∧ ∨ ∧ configuration.committed.index = proposal[i].rollback.index
                     ∧ LET changeIndex    ≜ proposal[proposal[i].rollback.index].rollback.index
                           changeValues   ≜ proposal[proposal[i].rollback.index].rollback.values
                           rollbackValues ≜ proposal[proposal[i].rollback.index].change.values
                          Record the changes required to roll back the target proposal and the index to
                          which the configuration is being rolled back.
                       IN  ∧ proposal′ = [proposal EXCEPT ![i].change = [index  ↦ changeIndex,
                                                                          values ↦ changeValues],
                                                          ![i].change = [index  ↦ proposal[i].chang
                                                                          values ↦ changeValues],
                                                          ![i].state  = ProposalComplete]
                    If the Rollback target is not the most recent change to the configuration,
                    fail validation for the proposal.
                   ∨ ∧ configuration.committed.index ≠ proposal[i].rollback.index
                     ∧ proposal′ = [proposal EXCEPT ![i].state = ProposalFailed]
                If a Rollback proposal is attempting to roll back another Rollback,
                fail validation for the proposal.
               ∨ ∧ proposal[proposal[i].rollback.index].type = ProposalRollback
                 ∧ proposal′ = [proposal EXCEPT ![i].state = ProposalFailed]
            ∧ UNCHANGED ⟨configuration, target⟩
          ∨ ∧ proposal[i].state = ProposalComplete
            ∧ proposal′ = [proposal EXCEPT ![i].phase = ProposalCommit,
                                           ![i].state = ProposalInProgress]
            ∧ UNCHANGED ⟨configuration, target⟩
          When a proposal is marked failed, set the configuration index to the proposal
          index to unblock subsequent proposals.
          ∨ ∧ proposal[i].state = ProposalFailed
            ∧ configuration′ = [configuration EXCEPT !.index = i]
            ∧ UNCHANGED ⟨proposal, target⟩
      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.index = i − 1
            ∧ configuration′ = [configuration EXCEPT !.committed.values = proposal[i].change.values,
                                                     !.committed.index  = proposal[i].change.index,
                                                     !.index            = i]
            ∧ proposal′ = [proposal EXCEPT ![i].state = ProposalComplete]
            ∧ UNCHANGED ⟨target⟩
          ∨ ∧ proposal[i].state = ProposalComplete
            ∧ 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.
$\lor$ $\land$ $proposal[i].phase = ProposalApply$
        If the node has no connection to the target, the proposal will be put in the
        pending state, otherwise the proposal will be in-progress until the changes
        can either be applied or fail.
   $\land$ $\lor$ $\land$ $proposal[i].state = ProposalPending$
         $\land$ $conn[n].state = Connected$
         $\land$ $proposal' = [proposal$ EXCEPT $![i].state = ProposalInProgress]$
         $\land$ UNCHANGED $\langle configuration, target \rangle$
      $\lor$ $\land$ $proposal[i].state\ \ = ProposalInProgress$
         $\land$ $mastership.master = n$
         $\land$ $conn[n].state = Disconnected$
         $\land$ $proposal' = [proposal$ EXCEPT $![i].state = ProposalPending]$
         $\land$ UNCHANGED $\langle configuration, target \rangle$
      $\lor$ $\land$ $proposal[i].state\ \ = ProposalInProgress$
         $\land$ $mastership.master = n$
         $\land$ $conn[n].state = Connected$
         $\land$ $target.state = Alive$
          Verify the applied index is the previous proposal index to ensure
          changes are applied to the target in order.
         $\land$ $configuration.applied.index = i - 1$
          Verify the applied term is the current mastership term to ensure the
          configuration has been synchronized following restarts.
         $\land$ $configuration.applied.term = mastership.term$
          Model successful and failed target update requests.
         $\land$ $\lor$ $\land$ $target' = [target$ EXCEPT $!.values = proposal[i].change.values]$
               $\land$ $configuration' = [configuration$ EXCEPT
                                    $!.applied.index\ \ = i,$
                                    $!.applied.values\ = proposal[i].change.values$
                                        @@ $configuration.applied.values]$
               $\land$ $proposal' = [proposal$ EXCEPT $![i].state = ProposalComplete]$
             If the proposal could not be applied, update the configuration's applied index
             and mark the proposal Failed.
            $\lor$ $\land$ $configuration' = [configuration$ EXCEPT $!.applied.index = i]$
               $\land$ $proposal' = [proposal$ EXCEPT $![i].state = ProposalFailed]$
               $\land$ UNCHANGED $\langle target \rangle$
$\lor$ $\land$ $proposal[i].phase = ProposalAbort$
   $\land$ $proposal[i].state\ = ProposalInProgress$
        If the configuration index is less than the proposal index, the proposal has
        not been committed, so it can be aborted without any additional changes required.
   $\land$ $\lor$ $\land$ $configuration.index = i - 1$
         $\land$ $configuration' = [configuration$ EXCEPT $!.index = i]$
         $\land$ $proposal' = [proposal$ EXCEPT $![i].state = ProposalComplete]$
      $\land$ UNCHANGED $\langle target \rangle$
    If the proposal has already been committed to the configuration but hasn't yet
    been applied to the target, we need to finish applying the proposal and fail
    the abort attempt.
    $\lor$ $\land$ $configuration.index \geq i$
      $\land$ $configuration.applied.index = i - 1$
      $\land$ $configuration.applied.term = mastership.term$
      $\land$ $mastership.master = n$
      $\land$ $conn[n].state = Connected$
      $\land$ $target.state = Alive$
       Model successful and failed target update requests.
      $\land$ $\lor$ $\land$ $target' = [target$ EXCEPT $!.values = proposal[i].change.values]$
           $\land$ $configuration' = [configuration$ EXCEPT
                                  $!.applied.index \;= i,$
                                  $!.applied.values = proposal[i].change.values$
                                      $@@ \; configuration.applied.values]$
           $\land$ $proposal' = [proposal$ EXCEPT $![i].state = ProposalComplete]$
         If the proposal could not be applied, update the configuration's applied index
         and mark the proposal Failed.
         $\lor$ $\land$ $configuration' = [configuration$ EXCEPT $!.applied.index = i]$
           $\land$ $proposal' = [proposal$ EXCEPT $![i].state = ProposalFailed]$
           $\land$ UNCHANGED $\langle target \rangle$
 $\land$ UNCHANGED $\langle mastership, conn \rangle$
```

Formal specification, constraints, and theorems.

```
$InitProposal \triangleq$
   $\land$ $proposal = [$
        $i \in \{\} \;\mapsto [$
          $phase \mapsto ProposalInitialize,$
          $state \;\mapsto ProposalInProgress]]$
   $\land$ $Trace!Init$

$NextProposal \triangleq$
   $\lor$ $\exists\, n \in Node :$
        $\exists\, i \in$ 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*