```
- Module Proposal -
EXTENDS Configuration, Mastership
INSTANCE Naturals
INSTANCE FiniteSets
LOCAL INSTANCE TLC
CONSTANT NumProposals
Assume NumProposals \in Nat
 Transaction type constants
CONSTANTS
   Proposal Change,
   ProposalRollback
 Phase constants
CONSTANTS
   Proposal Commit,\\
   Proposal Apply
 Status constants
CONSTANTS
   ProposalPending,
   Proposal In Progress,
   Proposal Complete,
   Proposal Failed \\
Constant LogProposal
Assume LogProposal \in Boolean
 A record of per-target proposals
Variable proposal
LOCAL CurrentState \triangleq [
                \mapsto [i \in \{i \in DOMAIN \ proposal : proposal[i].state \neq Nil\} \mapsto proposal[i]],
   configuration \mapsto configuration,
   target
                 \mapsto target,
   master ship \\
                 \mapsto mastership,
   nodes
                 \mapsto node
LOCAL SuccessorState \triangleq [
```

```
\mapsto [i \in \{i \in DOMAIN \ proposal' : proposal'[i].state \neq Nil\} \mapsto proposal'[i]],
    configuration \mapsto configuration',
    target
                     \mapsto target',
                     \mapsto mastership',
   mastership
   nodes
                     \mapsto node'
LOCAL Loq \stackrel{\triangle}{=} INSTANCE Loq WITH
                       \leftarrow "Proposal.log".
    CurrentState \leftarrow CurrentState,
    SuccessorState \leftarrow SuccessorState,
    Enabled
                       \leftarrow LogProposal
```

Commit a change to the configuration.

A change can be committed once all prior changes have been committed.

If a prior change is being rolled back, the rollback must complete before the change can be committed. Changes must be committed in sequential order.

Once a change commit is in progress, the change must be committed or failed before it can be applied or rolled back.

```
CommitChange(n, i) \triangleq
   \land \lor \land proposal[i].change.status = ProposalPending
          To commit a change, the commit index must be the prior index,
          and the target commit index must match the commit revision.
         \land configuration.commit.index = i - 1
         \land configuration.commit.revision = configuration.commit.target
         \land configuration' = [configuration \ EXCEPT \ !.commit.index = i,
                                                            !.commit.target = i
         \land proposal' = [proposal \ EXCEPT \ ![i].change.status = ProposalInProgress]
      \lor \land proposal[i].change.status = ProposalInProgress
          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.
         \land \lor \text{LET } rollbackRevision \stackrel{\triangle}{=} configuration.commit.revision
                    rollbackValues \stackrel{\triangle}{=} [p \in DOMAIN \ proposal[i].change.values \mapsto
                                                If p \in \text{DOMAIN} configuration.commit.values then
                                                   configuration.commit.values[p]
                                                ELSE
                                                   [index \mapsto 0, value \mapsto Nil]
                                         \stackrel{\triangle}{=} [p \in \text{DOMAIN } proposal[i].change.values \mapsto
                     change Values
                                               proposal[i].change.values[p]@@[index \mapsto i]]
```

 $\land configuration' = [configuration \ EXCEPT \ !.commit.revision = i,$ 

![i].change.phase

!.commit.values = change Values

= change Values,

= ProposalApply,

 $\land proposal' = [proposal \ \texttt{EXCEPT} \ ![i].change.values$ 

```
![i].change.status
                                                                                   = ProposalPending,
                                                          ![i].rollback.revision = rollbackRevision,
                                                          ![i].rollback.values
                                                                                   = rollbackValues
            \lor \land configuration' = [configuration \ EXCEPT \ !.commit.revision = i]
               \land proposal' = [proposal \ EXCEPT \ ![i].change.status = ProposalFailed]
   \land UNCHANGED \langle target \rangle
 Apply a change to the target.
 A change can be applied once all prior changes have been applied.
 If a prior change failed being applied, it must be rolled back before
 any subsequent change can be applied.
ApplyChange(n, i) \triangleq
   \land \lor \land proposal[i].change.status = ProposalPending
          To apply a change, the apply index must be the prior index.
         \land configuration.apply.index = i - 1
         \land configuration.apply.revision = configuration.apply.target
         \land configuration' = [configuration \ EXCEPT \ !.apply.index = i,
                                                         !.apply.target = i
         \land proposal' = [proposal \ EXCEPT \ ![i].change.status = ProposalInProgress]
         \land UNCHANGED \langle target \rangle
      \lor \land proposal[i].change.status = ProposalInProgress
          Verify the applied term is the current mastership term to ensure the
          configuration has been synchronized following restarts.
         \land configuration.apply.term = mastership.term
          Verify the node's connection to the target.
         \land node[n].connected
         \land mastership.conn = node[n].incarnation
         \land target.running
         \land node[n].target = target.incarnation
          Model successful and failed target update requests.
         \land \lor \land target' = [target \ EXCEPT \ !.values = proposal[i].change.values @@ target.values]
               \land LET values \stackrel{\triangle}{=} proposal[i].change.values @@ configuration.apply.values
                 IN configuration' = [configuration EXCEPT !.apply.revision]
                                                                                            =i.
                                                                     !.apply.incarnation = target.incarnation,
                                                                                            = values
                                                                     !.apply.values
               \land proposal' = [proposal \ EXCEPT \ ![i].change.status = ProposalComplete]
             If the proposal could not be applied, mark it failed but do not update the
             last applied index. The proposal must be rolled back before new proposals
             can be applied to the configuration/target.
            \lor \land proposal' = [proposal \ EXCEPT \ ![i].change.status = ProposalFailed]
               \land UNCHANGED \langle configuration, target \rangle
```

Commit a rollback to the configuration.  $\,$ 

A change can be rolled back once all subsequent, non-pending changes have been rolled back.

```
CommitRollback(n, i) \triangleq
   \land \lor \land proposal[i].rollback.status = ProposalPending
         \land \lor \land configuration.commit.target = i
               \land configuration.commit.revision = i
               \land configuration' = [configuration \ EXCEPT \ !.commit.target = proposal[i].rollback.revision]
               \land proposal' = [proposal \ EXCEPT \ ![i].rollback.status = ProposalInProgress]
            \lor \land configuration.commit.target = i
               \land configuration.commit.revision < i
               \land CommitChange(n, i)
            \lor \land configuration.commit.target < i
               \land proposal' = [proposal \ EXCEPT \ ![i].rollback.status = ProposalComplete]
               \land UNCHANGED \langle configuration, target \rangle
      \lor \land proposal[i].rollback.status = ProposalInProgress
         \land LET revision \stackrel{\triangle}{=} proposal[i].rollback.revision
                 values \triangleq proposal[i].rollback.values
                 \land configuration' = [configuration EXCEPT !.commit.revision = revision,
           IN
                                                                   ||.commit.values|| = values||
                  \land proposal' = [proposal \ EXCEPT \ ![i].rollback.phase = ProposalApply,
                                                        ![i].rollback.status = ProposalPending]
   \land UNCHANGED \langle target \rangle
 Commit a rollback to the target.
 A change can be rolled back once all subsequent, non-pending changes have been
 rolled back.
ApplyRollback(n, i) \stackrel{\Delta}{=}
   \land \lor \land proposal[i].rollback.status = ProposalPending
         \land \lor \land configuration.apply.target = i
               \land configuration.apply.revision = i
               \land configuration' = [configuration \ EXCEPT \ !.apply.target = proposal[i].rollback.revision]
               \land proposal' = [proposal \ EXCEPT \ ![i].rollback.status = ProposalInProgress]
               \land UNCHANGED \langle target \rangle
            \lor \land configuration.apply.target = i
               \land \ configuration.apply.revision < i
               \land ApplyChange(n, i)
            \lor \land configuration.apply.target < i
               \land proposal' = [proposal \ EXCEPT \ ![i].rollback.status = ProposalComplete]
               \land UNCHANGED \langle configuration, target \rangle
      \lor \land proposal[i].rollback.status = ProposalInProgress
          Verify the applied term is the current mastership term to ensure the
          configuration has been synchronized following restarts.
         \land configuration.apply.term = mastership.term
          Verify the node's connection to the target.
         \land node[n].connected
         \land target.running
         \land target' = [target \ EXCEPT \ !.values = proposal[i].rollback.values @@ target.values]
```

```
configuration' = [configuration \ EXCEPT \ !.apply.revision = revision,]
                                                             !.apply.values = values
         \land proposal' = [proposal \ EXCEPT \ ![i].rollback.status = ProposalComplete]
ReconcileProposal(n, i) \triangleq
   \land mastership.master = n
   \land \lor \land proposal[i].state = ProposalChange
         \land \lor \land proposal[i].change.phase = ProposalCommit
               \land CommitChange(n, i)
            \lor \land proposal[i].change.phase = ProposalApply
               \land ApplyChange(n, i)
      \lor \land proposal[i].state = ProposalRollback
         \land \lor \land proposal[i].rollback.phase = ProposalCommit
              \land \ CommitRollback(n, \ i)
            \lor \land proposal[i].rollback.phase = ProposalApply
               \land ApplyRollback(n, i)
   \land UNCHANGED \langle mastership, node \rangle
Formal specification, constraints, and theorems.
InitProposal \triangleq
   \land Log!Init
   \land proposal = [
         i \in 1 \dots NumProposals \mapsto [
           state
                     \mapsto Nil,
           change \mapsto [
              values \mapsto [p \in \{\} \mapsto [index \mapsto 0, value \mapsto Nil]],
              phase \mapsto Nil,
              status \mapsto Nil].
           rollback \mapsto [
```

```
NextProposal \triangleq
```

 $\vee \exists n \in Nodes:$ 

 $\exists i \in \text{DOMAIN } proposal :$ 

 $revision \mapsto 0$ ,

 $\begin{array}{ll} phase & \mapsto Nil, \\ status & \mapsto Nil]]]$ 

 $Log!Action(ReconcileProposal(n, i), [node \mapsto n, index \mapsto i])$ 

 $values \mapsto [p \in \{\} \mapsto [index \mapsto 0, value \mapsto Nil]],$ 

<sup>\\*</sup> Modification History

<sup>\*</sup> Last modified Fri Apr 21 19:15:11 PDT 2023 by jhalterm