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 $InitState \triangleq [$
$proposals \mapsto proposal,$
$configurations \mapsto configuration,$
$targets \mapsto target,$
$masterships \mapsto mastership,$
$nodes \mapsto node]$

LOCAL $NextState \triangleq [$
$proposals \mapsto proposal',$
$configurations \mapsto configuration',$
$targets \mapsto target',$
$masterships \mapsto mastership',$
$nodes \mapsto node']$

```
LOCAL Trace ≜ INSTANCE Trace WITH
  Module    ← "Proposals",
  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 revision to which the rollback changes
                will roll back the configuration.
            ∧ ∨ LET rollbackRevision ≜ configuration.committed.revision
                    rollbackValues   ≜ [p ∈ DOMAIN proposal[i].change.values ↦
                                         IF p ∈ DOMAIN configuration.committed.values THEN
                                           configuration.committed.values[p]
                                         ELSE
                                           [delete ↦ TRUE]]
                    changeValues     ≜ [p ∈ DOMAIN proposal[i].change.values ↦
                                         proposal[i].change.values[p] @@ [index ↦ i]]
              IN  ∧ configuration′ = [configuration EXCEPT !.committed.revision = i,
                                                           !.committed.values   = changeValues]
                  ∧ proposal′ = [proposal EXCEPT ![i].change = [
                                                   revision ↦ i,
                                                   values   ↦ changeValues],
                                                 ![i].rollback = [
                                                   revision ↦ rollbackRevision,
                                                   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 lastest change made
to the configuration based on the configuration revision.

$\land \lor \land configuration.committed.revision = proposal[i].rollback.index$

Record the changes required to roll back the target proposal and the index to
which the configuration is being rolled back.

$\land$ LET $changeRevision \triangleq proposal[proposal[i].rollback.index].rollback.revision$
$changeValues \triangleq proposal[proposal[i].rollback.index].rollback.values$
IN $\land configuration' = [configuration$ EXCEPT $!.committed.revision = changeR$
$!.committed.values = changeV$
$\land proposal' = [proposal$ EXCEPT $![i].change = [$
$revision \mapsto changeRevision,$
$values \mapsto changeValues],$
$![i].state = ProposalComplete]$

If the Rollback target is not the most recent change to the configuration,
fail validation for the proposal.

$\lor \land configuration.committed.revision \neq proposal[i].rollback.index$
$\land proposal' = [proposal$ EXCEPT $![i].state = ProposalFailed]$
$\land$ UNCHANGED $\langle configuration \rangle$

If a Rollback proposal is attempting to roll back another Rollback,
fail validation for the proposal.

$\lor \land proposal[proposal[i].rollback.index].type = ProposalRollback$
$\land proposal' = [proposal$ EXCEPT $![i].state = ProposalFailed]$
$\land$ UNCHANGED $\langle configuration \rangle$

$\land$ UNCHANGED $\langle target \rangle$

Once the proposal is committed, update the configuration's commit index
and move to the apply phase.

$\lor \land proposal[i].state = ProposalComplete$
$\land configuration' = [configuration$ EXCEPT $!.committed.index = i]$
$\land proposal' = [proposal$ EXCEPT $![i].phase = ProposalApply,$
$![i].state = ProposalInProgress]$
$\land$ UNCHANGED $\langle target \rangle$

If the proposal fails, mark the configuration applied for the proposal index.

$\lor \land proposal[i].state = ProposalFailed$
$\land \lor \land configuration.committed.index = i - 1$
$\land configuration' = [configuration$ EXCEPT $!.committed.index = i]$
$\lor \land configuration.applied.index = i - 1$
$\land configuration' = [configuration$ EXCEPT $!.applied.index = i]$
$\land$ UNCHANGED $\langle proposal, target \rangle$

While in the Apply phase, apply the proposed changes to the target.

$\lor \land proposal[i].phase = ProposalApply$

For the proposal to be applied, the node must be connected to a running target.

$\land \lor \land proposal[i].state = ProposalInProgress$

```
                Verify the applied index is the previous proposal index to ensure
                changes are applied to the target in order.
              ∧ configuration.applied.index = i − 1
                Verify the applied term is the current mastership term to ensure the
                configuration has been synchronized following restarts.
              ∧ configuration.applied.term = mastership.term
                Verify the node's connection to the target.
              ∧ node[n].connected
              ∧ target.running
                Model successful and failed target update requests.
              ∧ ∨ ∧ target' = [target EXCEPT !.values = proposal[i].change.values]
                  ∧ LET revision ≜ proposal[i].change.revision
                        values   ≜ proposal[i].change.values @@ configuration.applied.values
                    IN  configuration' = [configuration EXCEPT !.applied.index    = i,
                                                               !.applied.revision = revision,
                                                               !.applied.values   = values]
                  ∧ proposal' = [proposal EXCEPT ![i].state = ProposalComplete]
                If the proposal could not be applied, update the configuration's applied index
                and mark the proposal Failed.
                ∨ ∧ proposal' = [proposal EXCEPT ![i].state = ProposalFailed]
                  ∧ UNCHANGED ⟨configuration, target⟩
            Once the proposal is applied, update the configuration's applied index.
            ∨ ∧ proposal[i].state = ProposalComplete
              ∧ configuration' = [configuration EXCEPT !.applied.index = i]
              ∧ UNCHANGED ⟨proposal, target⟩
            If the proposal fails, mark the configuration applied for the proposal index.
            ∨ ∧ proposal[i].state = ProposalFailed
              ∧ configuration.applied.index = i − 1
              ∧ configuration' = [configuration EXCEPT !.applied.index = i]
              ∧ UNCHANGED ⟨proposal, target⟩
        ∧ UNCHANGED ⟨mastership, node⟩
```

Formal specification, constraints, and theorems.

```
InitProposal ≜
    ∧ proposal = [
        i ∈ {} ↦ [
          type      ↦ ProposalChange,
          change    ↦ [
            revision ↦ 0,
            values   ↦ [p ∈ {} ↦ [index ↦ 0, value ↦ Nil, delete ↦ FALSE]]],
          rollback ↦ [
            index    ↦ 0,
            revision ↦ 0,
```

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

$$
\begin{array}{l}
NextProposal \triangleq \\
\quad \lor \exists\, n \in Node : \\
\quad\quad \exists\, i \in \text{DOMAIN } proposal : \\
\quad\quad\quad Trace!Step(ReconcileProposal(n,\ i),\ [node \mapsto n,\ index \mapsto i])
\end{array}
$$

* 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*