MODULE *Proposal*

INSTANCE *Naturals*

INSTANCE *FiniteSets*

INSTANCE *Sequences*

INSTANCE *TLC*

An empty constant
CONSTANT *Nil*

Phase constants
CONSTANTS
Change,
Rollback

Phase $\triangleq$
{*Change*,
Rollback}

Step constants
CONSTANTS
Commit,
Apply

Status constants
CONSTANTS
Pending,
InProgress,
Complete,
Failed

Status $\triangleq$
{*Nil*,
Pending,
InProgress,
Complete,
Failed}

The set of all nodes
CONSTANT *Node*

Variables defined by other modules.

```
VARIABLES
    configuration,
    mastership,
    conn,
    target

 A record of per-target proposals
VARIABLE proposal

 A sequence of configuration changes used for model checking.
VARIABLE history

TypeOK ≜
  ∀ i ∈ DOMAIN proposal :
    ∧ proposal[i].phase ∈ Phase
    ∧ proposal[i].change.commit ∈ Status
    ∧ proposal[i].change.apply ∈ Status
    ∧ ∀ p ∈ DOMAIN proposal[i].change.values :
        ∧ proposal[i].change.values[p].index ∈ Nat
        ∧ proposal[i].change.values[p].value ≠ Nil ⇒
            proposal[i].change.values[p].value ∈ STRING
    ∧ proposal[i].rollback.commit ∈ Status
    ∧ proposal[i].rollback.apply ∈ Status
    ∧ proposal[i].rollback.revision ∈ Nat
    ∧ ∀ p ∈ DOMAIN proposal[i].rollback.values :
        ∧ proposal[i].rollback.values[p].index ∈ Nat
        ∧ proposal[i].rollback.values[p].value ≠ Nil ⇒
            proposal[i].rollback.values[p].value ∈ STRING

LOCAL State ≜ [
  proposals     ↦ [i ∈ DOMAIN proposal ↦ proposal[i] @@ [index ↦ i]],
  configuration ↦ configuration,
  mastership    ↦ mastership,
  conns         ↦ conn,
  target        ↦ target]

LOCAL Transitions ≜
  LET
    proposals ≜ {i ∈ DOMAIN proposal' :
                   i ∈ DOMAIN proposal ⇒ proposal'[i] ≠ proposal[i]}
  IN
    [proposals ↦ [i ∈ proposals     ↦ proposal'[i] @@ [index ↦ i]]] @@
    (IF configuration' ≠ configuration THEN [configuration ↦ configuration'] ELSE ⟨⟩) @@
    (IF target' ≠ target THEN [target ↦ target'] ELSE ⟨⟩)

Test ≜ INSTANCE Test WITH
  File      ← "Proposal.log"
```

$CommitChange(n,\ i) \triangleq$

$\land\ proposal[i].change.commit = InProgress$

If the committed index does not match the proposal index, commit the change.

$\land\ \lor\ \land\ configuration.committed.index = i - 1$

If the change is valid, update the committed index, revision, and values.

$\land\ \lor\ \land\ configuration' = [configuration\ \text{EXCEPT}\ !.committed.index = i,$
$!.committed.revision = i,$
$!.committed.values = proposal[i].change.values\ @@$
$configuration.committed.va$

$\land\ history' = Append(history, [type \mapsto Change, phase \mapsto Commit, index \mapsto i])$

If the change is invalid, update only the committed index.

$\lor\ \land\ configuration' = [configuration\ \text{EXCEPT}\ !.committed.index = i]$
$\land\ \text{UNCHANGED}\ \langle history \rangle$

$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

If both the committed index and committed revision were updated, the proposal was successful.

$\lor\ \land\ configuration.committed.index = i$
$\land\ configuration.committed.revision = i$
$\land\ proposal' = [proposal\ \text{EXCEPT}\ ![i].change.commit = Complete]$
$\land\ \text{UNCHANGED}\ \langle configuration, history \rangle$

If the committed index was updated but the revision was not, the proposal failed validation.

$\lor\ \land\ configuration.committed.index = i$
$\land\ configuration.committed.revision \neq i$
$\land\ proposal' = [proposal\ \text{EXCEPT}\ ![i].change.commit = Failed]$
$\land\ \text{UNCHANGED}\ \langle configuration, history \rangle$

$\land\ \text{UNCHANGED}\ \langle target \rangle$

$ApplyChange(n,\ i) \triangleq$

$\land\ proposal[i].change.apply = InProgress$

If the applied index does not match the proposal index, apply the change.

$\land\ \lor\ \land\ configuration.applied.index = i - 1$
$\land\ configuration.state = Complete$
$\land\ configuration.term = mastership.term$
$\land\ conn[n].id = mastership.conn$
$\land\ conn[n].connected$
$\land\ target.running$

If the change can be applied, update the index, revision, and values.

$\land\ \lor\ \land\ target' = [target\ \text{EXCEPT}\ !.values = proposal[i].change.values\ @@\ target.values]$
$\land\ configuration' = [configuration\ \text{EXCEPT}\ !.applied.index = i,$
$!.applied.revision = i,$
$!.applied.values = proposal[i].change.values\ @@$
$configuration.applied.values]$

$\land\ history' = Append(history, [type \mapsto Change, phase \mapsto Apply, index \mapsto i])$

If the change is invalid, update only the applied index.

$\lor \land configuration' = [configuration \text{ EXCEPT } !.applied.index = i]$
$\land \text{UNCHANGED } \langle target, history \rangle$
$\land \text{UNCHANGED } \langle proposal \rangle$

If the applied index and revision both match the proposal index, the change was successful.

$\lor \land configuration.applied.index = i$
$\land configuration.applied.revision = i$
$\land proposal' = [proposal \text{ EXCEPT } ![i].change.apply = Complete]$
$\land \text{UNCHANGED } \langle configuration, target, history \rangle$

If the applied index matches the proposal index but the revision does not, the proposal failed.

$\lor \land configuration.applied.index = i$
$\land configuration.applied.revision \neq i$
$\land proposal' = [proposal \text{ EXCEPT } ![i].change.apply = Failed]$
$\land \text{UNCHANGED } \langle configuration, target, history \rangle$

$CommitRollback(n, i) \triangleq$

$\land proposal[i].rollback.commit = InProgress$

If the committed revision matches the proposal revision, roll back to the previous revision.

$\land \lor \land configuration.committed.revision = i$
$\land configuration' = [configuration \text{ EXCEPT } !.committed.revision = proposal[i].rollback.revision,$
$!.committed.values = proposal[i].rollback.values \mathbin{@@} configuration.committed.values]$
$\land history' = Append(history, [type \mapsto Rollback, phase \mapsto Commit, index \mapsto i])$
$\land \text{UNCHANGED } \langle proposal \rangle$

If the committed index matches the rollback index, complete the rollback.

$\lor \land configuration.committed.revision = proposal[i].rollback.revision$
$\land proposal' = [proposal \text{ EXCEPT } ![i].rollback.commit = Complete]$
$\land \text{UNCHANGED } \langle configuration, history \rangle$

$\land \text{UNCHANGED } \langle target \rangle$

$ApplyRollback(n, i) \triangleq$

$\land proposal[i].rollback.apply = InProgress$

If the applied revision matches the proposal revision, roll back to the previous revision.

$\land \lor \land configuration.applied.revision = i$
$\land configuration.state = Complete$
$\land configuration.term = mastership.term$
$\land conn[n].id = mastership.conn$
$\land conn[n].connected$
$\land target.running$
$\land target' = [target \text{ EXCEPT } !.values = proposal[i].rollback.values \mathbin{@@} target.values]$
$\land configuration' = [configuration \text{ EXCEPT } !.applied.revision = proposal[i].rollback.revision,$
$!.applied.values = proposal[i].rollback.values \mathbin{@@} configuration.applied.values]$
$\land history' = Append(history, [type \mapsto Rollback, phase \mapsto Apply, index \mapsto i])$
$\land \text{UNCHANGED } \langle proposal \rangle$

If the committed index matches the rollback index, complete the rollback.

$\lor \land configuration.committed.revision = proposal[i].rollback.revision$
$\land proposal' = [proposal \text{ EXCEPT } ![i].rollback.apply = Complete]$
$\land \text{UNCHANGED } \langle configuration, target, history \rangle$

Reconcile a proposal

$ReconcileProposal(n, i) \triangleq$
$\land i \in \text{DOMAIN } proposal$
$\land mastership.master = n$
$\land \lor CommitChange(n, i)$
$\lor ApplyChange(n, i)$
$\lor CommitRollback(n, i)$
$\lor ApplyRollback(n, i)$
$\land \text{UNCHANGED } \langle mastership, conn \rangle$