```
MODULE Config

INSTANCE Naturals

INSTANCE FiniteSets

INSTANCE Sequences

INSTANCE TLC

 An empty constant
CONSTANT Nil

 Transaction type constants
CONSTANTS
    Change,
    Rollback

 Transaction isolation constants
CONSTANTS
    ReadCommitted,
    Serializable

 Status constants
CONSTANTS
    Pending,
    Initializing,
    Initialized,
    Validating,
    Validated,
    Committing,
    Committed,
    Applying,
    Applied,
    Synchronizing,
    Synchronized,
    Persisted,
    Failed

Status ≜
    ⟨Initializing,
     Initialized,
     Validating,
     Validated,
     Committing,
     Committed,
```

$\quad Applying,$
$\quad Applied,$
$\quad Failed \rangle$

$\textsc{constants}$
$\quad Valid,$
$\quad Invalid$

$\textsc{constants}$
$\quad Success,$
$\quad Failure$

The set of all nodes

$\textsc{constant}\ Node$

Target is the set of all targets and their possible paths and values.

Example: $Target \triangleq [$
$\quad target1 \mapsto [persistent \mapsto \textsc{false}, values \mapsto [$
$\qquad path1 \mapsto \{\text{“}value1\text{”}, \text{“}value2\text{”}\},$
$\qquad path2 \mapsto \{\text{“}value2\text{”}, \text{“}value3\text{”}\}]],$
$\quad target2 \mapsto [persistent \mapsto \textsc{true}, values \mapsto [$
$\qquad path2 \mapsto \{\text{“}value3\text{”}, \text{“}value4\text{”}\},$
$\qquad path3 \mapsto \{\text{“}value4\text{”}, \text{“}value5\text{”}\}]]]$

$\textsc{constant}\ Target$

$Phase(s) \triangleq \textsc{choose}\ i \in \textsc{domain}\ Status : Status[i] = s$

$\textsc{assume}\ Nil \in \textsc{string}$

$\textsc{assume}\ Pending \in \textsc{string}$
$\textsc{assume}\ Initializing \in \textsc{string}$
$\textsc{assume}\ Initialized \in \textsc{string}$
$\textsc{assume}\ Validating \in \textsc{string}$
$\textsc{assume}\ Validated \in \textsc{string}$
$\textsc{assume}\ Committing \in \textsc{string}$
$\textsc{assume}\ Committed \in \textsc{string}$
$\textsc{assume}\ Applying \in \textsc{string}$
$\textsc{assume}\ Applied \in \textsc{string}$
$\textsc{assume}\ Synchronizing \in \textsc{string}$
$\textsc{assume}\ Synchronized \in \textsc{string}$
$\textsc{assume}\ Persisted \in \textsc{string}$
$\textsc{assume}\ Failed \in \textsc{string}$

$\textsc{assume}\ \land IsFiniteSet(Node)$
$\qquad\qquad \land \forall n \in Node :$
$\qquad\qquad\quad \land n \notin \textsc{domain}\ Target$
$\qquad\qquad\quad \land n \in \textsc{string}$

```
ASSUME  ∧ ∀ t ∈ DOMAIN  Target :
            ∧ t ∉ Node
            ∧ t ∈ STRING
            ∧ Target[t].persistent ∈ BOOLEAN
            ∧ ∀ p ∈ DOMAIN  Target[t].values :
                IsFiniteSet(Target[t].values[p])
```

Configuration update/rollback requests are tracked and processed through two data types. Transactions represent the lifecycle of a single configuration change request and are stored in an append-only log. Configurations represent the desired configuration of a *gNMI* target based on the aggregate of relevant changes in the Transaction log.

```
TYPE Type  ::= type ∈
  {Change,
   Rollback}

TYPE Status ::= status ∈
  {Pending,
   Initializing,
   Initialized,
   Validating,
   Validated,
   Committing,
   Committed,
   Applying,
   Applied,
   Synchronizing,
   Synchronized,
   Persisted,
   Failed}

TYPE Transaction  ≜  [
  type      ::= type ∈  Type,
  index     ::= index ∈ Nat,
  isolation ::= isolation ∈ {IsolationDefault, IsolationSerializable}
  values ::=  [
    target ∈ SUBSET  (DOMAIN  Target) ↦ [ path ∈ SUBSET  (DOMAIN  Target[target].values) ↦
      [
        value ::= value ∈ STRING,
        delete ::= delete ∈ BOOLEAN ]]],
  rollback ::= index ∈ Nat,
  targets ::= targets ∈ SUBSET  (DOMAIN  Target)
  status ::= status ∈ Status]

TYPE Proposal  ≜  [
  type          ::= type ∈  Type,
  index         ::= index ∈ Nat,
  values        ::=  [ path ∈ SUBSET  (DOMAIN  Target[target].values) ↦ [
      value ::= value ∈ STRING,
      delete ::= delete ∈ BOOLEAN ]],
  rollback        ::= index ∈ Nat,
  dependencyIndex ::= dependencyIndex ∈ Nat,
  rollbackIndex ::= rollbackIndex ∈ Nat,
  rollbackValues ::= [ path ∈ SUBSET (DOMAIN Target[target].values) ↦ [
       value ::= value ∈ STRING,
       delete ::= delete ∈ BOOLEAN ]],
  status       ::= status ∈ Status]

TYPE ConfigurationStatus ::= status ∈
  {ConfigurationUnknown,
   ConfigurationSynchronizing,
   ConfigurationSynchronized,
   ConfigurationPersisted,
   ConfigurationFailed}

TYPE Configuration ≜ [
  id            ::= id ∈ STRING,
  target        ::= target ∈ STRING,
  values        ::= [ path ∈ SUBSET (DOMAIN Target[target]) ↦ [
       value ::= value ∈ STRING,
       index ::= index ∈ Nat,
       deleted ::= delete ∈ BOOLEAN ]],
  configIndex ::= configIndex ∈ Nat,
  configTerm     ::= configTerm ∈ Nat,
  proposedIndex ::= proposedIndex ∈ Nat,
  committedIndex ::= committedIndex ∈ Nat,
  appliedIndex ::= appliedIndex ∈ Nat,
  appliedTerm ::= appliedTerm ∈ Nat,
  appliedValues ::= [ path ∈ SUBSET (DOMAIN Target[target]) ↦ [
       value ::= value ∈ STRING,
       index ::= index ∈ Nat,
       deleted ::= delete ∈ BOOLEAN ]],
  status ::= status ∈ Status]
```

A transaction log. Transactions may either request a set of changes to a set of targets or rollback a prior change.

VARIABLE *transaction*

A record of per-target proposals

VARIABLE *proposal*

A record of per-target configurations

VARIABLE *configuration*

A record of target states

VARIABLE *target*

A record of target masterships

VARIABLE *mastership*

$$vars \triangleq \langle transaction, proposal, configuration, mastership, target \rangle$$

This section models *mastership* for the configuration service.

Mastership is used primarily to track the lifecycle of individual configuration targets and react to state changes on the southbound. Each target is assigned a master from the *Node* set, and masters can be unset when the target disconnects.

Set node n as the master for target t

$$\begin{array}{l}
\mathit{SetMaster}(n,\ t) \triangleq \\
\quad \land\ \mathit{mastership}[t].\mathit{master} \neq n \\
\quad \land\ \mathit{mastership}' = [\mathit{mastership}\ \textsc{except}\ ![t].\mathit{term} \quad = \mathit{mastership}[t].\mathit{term} + 1, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![t].\mathit{master} = n] \\
\quad \land\ \textsc{unchanged}\ \langle \mathit{transaction},\ \mathit{proposal},\ \mathit{configuration},\ \mathit{target} \rangle
\end{array}$$

$$\begin{array}{l}
\mathit{UnsetMaster}(t) \triangleq \\
\quad \land\ \mathit{mastership}[t].\mathit{master} \neq \mathit{Nil} \\
\quad \land\ \mathit{mastership}' = [\mathit{mastership}\ \textsc{except}\ ![t].\mathit{master} = \mathit{Nil}] \\
\quad \land\ \textsc{unchanged}\ \langle \mathit{transaction},\ \mathit{proposal},\ \mathit{configuration},\ \mathit{target} \rangle
\end{array}$$

This section models configuration changes and rollbacks. Changes are appended to the transaction log and processed asynchronously.

$$\begin{array}{l}
\mathit{Value}(s,\ t,\ p) \triangleq \\
\quad \textsc{let}\ \mathit{value} \triangleq \textsc{choose}\ v \in s : v.\mathit{target} = t \land v.\mathit{path} = p \\
\quad \textsc{in} \\
\qquad [\mathit{value} \ \mapsto \mathit{value}.\mathit{value}, \\
\qquad\ \ \mathit{delete} \mapsto \mathit{value}.\mathit{delete}]
\end{array}$$

$$\begin{array}{l}
\mathit{Paths}(s,\ t) \triangleq \\
\quad [p \in \{v.\mathit{path} : v \in \{v \in s : v.\mathit{target} = t\}\} \mapsto \mathit{Value}(s,\ t,\ p)]
\end{array}$$

$$\begin{array}{l}
\mathit{Changes}(s) \triangleq \\
\quad [t \in \{v.\mathit{target} : v \in s\} \mapsto \mathit{Paths}(s,\ t)]
\end{array}$$

$$\begin{array}{l}
\mathit{ValidValues}(t,\ p) \triangleq \\
\quad \textsc{union}\ \{\{[\mathit{value} \mapsto v,\ \mathit{delete} \mapsto \textsc{false}] : v \in \mathit{Target}[t].\mathit{values}[p]\},\ \{[\mathit{value} \mapsto \mathit{Nil},\ \mathit{delete} \mapsto \textsc{true}]\}\}
\end{array}$$

$$\begin{array}{l}
\mathit{ValidPaths}(t) \triangleq \\
\quad \textsc{union}\ \{\{v\ @@\ [\mathit{path} \mapsto p] : v \in \mathit{ValidValues}(t,\ p)\} : p \in \textsc{domain}\ \mathit{Target}[t].\mathit{values}\}
\end{array}$$

$$\begin{array}{l}
\mathit{ValidTargets} \triangleq \\
\quad \textsc{union}\ \{\{p\ @@\ [\mathit{target} \mapsto t] : p \in \mathit{ValidPaths}(t)\} : t \in \textsc{domain}\ \mathit{Target}\}
\end{array}$$

The set of all valid sets of changes to all targets and their paths.
The set of possible changes is computed from the *Target* model value.

$$\begin{array}{l}
\mathit{ValidChanges} \triangleq \\
\quad \textsc{let}\ \mathit{changeSets} \triangleq \{s \in \textsc{subset}\ \mathit{ValidTargets} : \\
\qquad\qquad\qquad\qquad\qquad \forall\, t \in \textsc{domain}\ \mathit{Target} \quad :
\end{array}$$

```
                              ∀ p ∈ DOMAIN Target[t].values :
                                Cardinality({v ∈ s : v.target = t ∧ v.path = p}) ≤ 1}
  IN
     {Changes(s) : s ∈ changeSets}
```

The next available index in the transaction log.
This is computed as the max of the existing indexes in the log to
allow for changes to the log (*e.g.* log compaction) to be modeled.

```
NextIndex ≜
  IF DOMAIN transaction = {} THEN
     1
   ELSE
     LET i ≜ CHOOSE i ∈ DOMAIN transaction :
             ∀ j ∈ DOMAIN transaction : i ≥ j
     IN   i + 1
```

Add a set of changes 'c' to the transaction log

```
RequestChange(c) ≜
  ∧ ∃ isolation ∈ {ReadCommitted, Serializable} :
      ∧ transaction′ = transaction @@ (NextIndex :> [type      ↦ Change,
                                                     index     ↦ NextIndex,
                                                     isolation ↦ isolation,
                                                     values    ↦ c,
                                                     targets   ↦ {},
                                                     status    ↦ Initializing])
  ∧ UNCHANGED ⟨proposal, configuration, mastership, target⟩
```

Add a rollback of transaction 't' to the transaction log

```
RequestRollback(t) ≜
  ∧ ∃ isolation ∈ {ReadCommitted, Serializable} :
      ∧ transaction′ = transaction @@ (NextIndex :> [type      ↦ Rollback,
                                                     index     ↦ NextIndex,
                                                     isolation ↦ isolation,
                                                     rollback  ↦ t,
                                                     targets   ↦ {},
                                                     status    ↦ Initializing])
  ∧ UNCHANGED ⟨proposal, configuration, mastership, target⟩
```

This section models the Transaction log reconciler.

Transactions come in two flavors : − *Change* transactions contain a set of changes to be applied to a set of *targets* − *Rollback* transactions reference a prior change transaction to be reverted to the previous state

Transacations proceed through a series of phases:
* Initialize - create and link Proposals

* Validate - validate changes and rollbacks
* *Commit* – commit changes to Configurations
* Apply - commit changes to Targets

```
 Reconcile a transaction
$ReconcileTransaction(n, i) \triangleq$
        Initializing is the only transaction phase that's globally serialized.
        While in the *Initializing* phase, the reconciler checks whether the
        prior transaction has been *Initialized* before creating Proposals in
        the Initialize phase. Once all of the transaction's proposals have
        been *Initialized*, the transaction will be marked *Initialized*. If any
        proposal is *Failed*, the transaction will be marked *Failed* as well.
    $\land \lor \land transaction[i].status = Initializing$
           Serialize transaction initialization
           $\land i - 1 \in \text{DOMAIN } transaction \Rightarrow$
                   $Phase(transaction[i-1].status) > Phase(Initializing)$
           If the transaction's targets are not yet set, create proposals
           and add targets to the transaction state.
           $\land \lor \land transaction[i].targets = \{\}$
                    If the transaction is a change, the targets are taken
                    from the change values.
                 $\land \lor \land transaction[i].type = Change$
                          $\land transaction' = [transaction \text{ EXCEPT } ![i].targets = \text{DOMAIN } transaction[i].values]$
                          $\land proposal' = [t \in \text{DOMAIN } proposal \mapsto$
                                $\text{IF } t \in \text{DOMAIN } transaction[i].values \text{ THEN}$
                                   $proposal[t] @@ (i :> [type \mapsto Change,$
                                                          $index \mapsto i,$
                                                          $values \mapsto transaction[i].values[t],$
                                                          $dependencyIndex \mapsto 0,$
                                                          $rollbackIndex \mapsto 0,$
                                                          $rollbackValues \mapsto \langle\rangle,$
                                                          $status \mapsto Initializing])$
                                $\text{ELSE}$
                                   $proposal[t]]$
                    If the transaction is a rollback, the targets affected are
                    the targets of the change transaction being rolled back.
                    $\lor \land transaction[i].type = Rollback$
                       $\land \lor \land transaction[i].rollback \in \text{DOMAIN } transaction$
                             $\land transaction[transaction[i].rollback].type = Change$
                             $\land transaction' = [transaction \text{ EXCEPT } ![i].targets =$
                                                     $\text{DOMAIN } transaction[transaction[i].rollback].values]$
                             $\land proposal' = [t \in \text{DOMAIN } proposal \mapsto$
                                   $\text{IF } t \in \text{DOMAIN } transaction[transaction[i].rollback].values \text{ THEN}$
                                      $proposal[t] @@ (i :> [type \mapsto Rollback,$
                                                             $index \mapsto i,$
                                                             $rollback \mapsto transaction[i].rollback,$
```

```
                                                  $dependencyIndex \mapsto 0,$
                                                  $rollbackIndex \mapsto 0,$
                                                  $rollbackValues \mapsto \langle\rangle,$
                                                  $status \mapsto Initializing])$
                                    ELSE
                                      $proposal[t]]$
                               $\lor \land \lor \land transaction[i].rollback \in \text{DOMAIN } transaction$
                                          $\land transaction[transaction[i].rollback].type = Rollback$
                                        $\lor transaction[i].rollback \notin \text{DOMAIN } transaction$
                                  $\land transaction' = [transaction \text{ EXCEPT } ![i].status = Failed]$
                                  $\land \text{UNCHANGED } \langle proposal\rangle$
                         $\lor \land transaction[i].targets \neq \{\}$
                            If all proposals have been Initialized, mark the transaction Initialized.
                           $\land \lor \land \forall t \in transaction[i].targets : proposal[t][i].status = Initialized$
                                 $\land transaction' = [transaction \text{ EXCEPT } ![i].status = Initialized]$
                                 $\land \text{UNCHANGED } \langle proposal\rangle$
                            If any proposal has been Failed, mark the transaction Failed.
                              $\lor \land \exists t \in transaction[i].targets : proposal[t][i].status = Failed$
                                 $\land transaction' = [transaction \text{ EXCEPT } ![i].status = Failed]$
                                 $\land \text{UNCHANGED } \langle proposal\rangle$
                      Once the transaction has been Initialized, proceed to the Validate phase.
                      If any of the transaction's proposals depend on a Serializable transaction,
                      verify the dependency has been Validated to preserve serializability before
                      moving the transaction to the Validate phase.
                      $\lor \land transaction[i].status = Initialized$
                         $\land \forall t \in transaction[i].targets :$
                             $proposal[t][i].dependencyIndex \neq 0 \Rightarrow$
                                 $(transaction[proposal[t][i].dependencyIndex].isolation = Serializable \Rightarrow$
                                     $Phase(transaction[proposal[t][i].dependencyIndex].status) \geq Phase(Validated))$
                         $\land transaction' = [transaction \text{ EXCEPT } ![i].status = Validating]$
                         $\land \text{UNCHANGED } \langle proposal\rangle$
                      $\lor \land transaction[i].status = Validating$
                          Move the transaction's proposals to the Validating state
                         $\land \lor \land \exists t \in transaction[i].targets : Phase(proposal[t][i].status) < Phase(Validating)$
                               $\land proposal' = [t \in \text{DOMAIN } proposal \mapsto$
                                                  IF $t \in transaction[i].targets$ THEN
                                                     $[proposal[t] \text{ EXCEPT } ![i].status = Validating]$
                                                   ELSE
                                                     $proposal[t]]$
                               $\land \text{UNCHANGED } \langle transaction\rangle$
                          If all proposals have been Validated, mark the transaction Validated.
                            $\lor \land \forall t \in transaction[i].targets : proposal[t][i].status = Validated$
                               $\land transaction' = [transaction \text{ EXCEPT } ![i].status = Validated]$
                               $\land \text{UNCHANGED } \langle proposal\rangle$
                          If any proposal has been Failed, mark the transaction Failed.
```

```
          $\lor \land \exists t \in transaction[i].targets : proposal[t][i].status = Failed$
             $\land transaction' = [transaction \text{ EXCEPT } ![i].status = Failed]$
             $\land \text{UNCHANGED } \langle proposal \rangle$
 Once the transaction has been Validated, proceed to the Commit phase.
 If any of the transaction's proposals depend on a Serializable transaction,
 verify the dependency has been Committed to preserve serializability before
 moving the transaction to the Commit phase.
$\lor \land transaction[i].status = Validated$
   $\land \forall t \in transaction[i].targets :$
        $proposal[t][i].dependencyIndex \neq 0 \Rightarrow$
           $(transaction[proposal[t][i].dependencyIndex].isolation = Serializable \Rightarrow$
              $Phase(transaction[proposal[t][i].dependencyIndex].status) \geq Phase(Committed))$
   $\land transaction' = [transaction \text{ EXCEPT } ![i].status = Committing]$
   $\land \text{UNCHANGED } \langle proposal \rangle$
$\lor \land transaction[i].status = Committing$
      Move the transaction's proposals to the Committing state
   $\land \lor \land \exists t \in transaction[i].targets : Phase(proposal[t][i].status) < Phase(Committing)$
         $\land proposal' = [t \in \text{DOMAIN } proposal \mapsto$
                          $\text{IF } t \in transaction[i].targets \text{ THEN}$
                             $[proposal[t] \text{ EXCEPT } ![i].status = Committing]$
                           $\text{ELSE}$
                             $proposal[t]]$
         $\land \text{UNCHANGED } \langle transaction \rangle$
      If all proposals have been Committed, mark the transaction Committed.
      $\lor \land \forall t \in transaction[i].targets : proposal[t][i].status = Committed$
         $\land transaction' = [transaction \text{ EXCEPT } ![i].status = Committed]$
         $\land \text{UNCHANGED } \langle proposal \rangle$
 Once the transaction has been Committed, proceed to the Apply phase.
 If any of the transaction's proposals depend on a Serializable transaction,
 verify the dependency has been Applied to preserve serializability before
 moving the transaction to the Apply phase.
$\lor \land transaction[i].status = Committed$
   $\land \forall t \in transaction[i].targets :$
        $proposal[t][i].dependencyIndex \neq 0 \Rightarrow$
           $(transaction[proposal[t][i].dependencyIndex].isolation = Serializable \Rightarrow$
              $Phase(transaction[proposal[t][i].dependencyIndex].status) \geq Phase(Applied))$
   $\land transaction' = [transaction \text{ EXCEPT } ![i].status = Applying]$
   $\land \text{UNCHANGED } \langle proposal \rangle$
$\lor \land transaction[i].status = Applying$
      Move the transaction's proposals to the Applying state
   $\land \lor \land \exists t \in transaction[i].targets : Phase(proposal[t][i].status) < Phase(Applying)$
         $\land proposal' = [t \in \text{DOMAIN } proposal \mapsto$
                          $\text{IF } t \in transaction[i].targets \text{ THEN}$
                             $[proposal[t] \text{ EXCEPT } ![i].status = Applying]$
                           $\text{ELSE}$
```

```
                              proposal[t]]
           ∧ UNCHANGED ⟨transaction⟩
          If all proposals have been Applied, mark the transaction Applied.
        ∨ ∧ ∀ t ∈ transaction[i].targets : proposal[t][i].status = Applied
          ∧ transaction′ = [transaction EXCEPT ![i].status = Applied]
          ∧ UNCHANGED ⟨proposal⟩
          If any proposal has been Failed, mark the transaction Failed.
        ∨ ∧ ∃ t ∈ transaction[i].targets : proposal[t][i].status = Failed
          ∧ transaction′ = [transaction EXCEPT ![i].status = Failed]
          ∧ UNCHANGED ⟨proposal⟩
  ∧ UNCHANGED ⟨configuration, mastership, target⟩

Reconcile a proposal
ReconcileProposal(n, t, i) ≜
  ∧ ∨ ∧ proposal[t][i].status = Initializing
      ∧ proposal′ = [proposal EXCEPT ![t] = [proposal[t] EXCEPT
           ![i] = [status          ↦ Initialized,
                   dependencyIndex ↦ configuration[t].proposedIndex] @@ proposal[t][i]]]
      ∧ configuration′ = [configuration EXCEPT ![t].proposedIndex = i]
      ∧ UNCHANGED ⟨target⟩
    While in the Validating state, 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[t][i].status = Validating
      ∧ configuration[t].committedIndex = proposal[t][i].dependencyIndex
        For Change proposals validate the set of requested changes.
      ∧ ∨ ∧ proposal[t][i].type = Change
          ∧ LET rollbackIndex ≜ configuration[t].configIndex
                rollbackValues ≜ [p ∈ DOMAIN proposal[t][i].values ↦
                                     IF p ∈ DOMAIN configuration[t].values THEN
                                        configuration[t].values[p]
                                      ELSE
                                        [value  ↦ Nil,
                                         delete ↦ TRUE]]
              Model validation successes and failures with Valid and Invalid results.
            IN  ∃ r ∈ {Valid, Invalid} :
                  If the Change is Valid, record the changes required to roll
                  back the proposal and the index to which the rollback changes
                  will roll back the configuration.
                  ∨ ∧ r = Valid
                    ∧ proposal′ = [proposal EXCEPT ![t] = [
                                     proposal[t] EXCEPT ![i].rollbackIndex  = rollbackIndex,
                                                        ![i].rollbackValues = rollbackValues,
                                                        ![i].status         = Validated]]
                    ∧ UNCHANGED ⟨configuration⟩
              $\lor \land r = Invalid$
                $\land configuration' = [configuration \text{ EXCEPT } ![t].committedIndex = i]$
                $\land proposal' = [proposal \text{ EXCEPT } ![t] = [$
                               $proposal[t] \text{ EXCEPT } ![i].status = Failed]]$
     For Rollback proposals, validate the rollback changes which are
     proposal being rolled back.
    $\lor \land proposal[t][i].type = Rollback$
         Rollbacks can only be performed on Change type proposals.
       $\land \lor \land proposal[t][proposal[t][i].rollback].type = Change$
              Only roll back the change if it's the lastest change made
              to the configuration based on the configuration index.
           $\land \lor \land configuration[t].configIndex = proposal[t][i].rollback$
               $\land \text{LET } rollbackIndex \triangleq proposal[t][proposal[t][i].rollback].rollbackIndex$
                     $rollbackValues \triangleq proposal[t][proposal[t][i].rollback].rollbackValues$
                 $\text{IN } \exists r \in \{Valid, Invalid\} :$
                       If the Rollback is Valid, record the changes required to
                       roll back the target proposal and the index to which the
                       configuration is being rolled back.
                       $\lor \land r = Valid$
                         $\land proposal' = [proposal \text{ EXCEPT } ![t] = [$
                             $proposal[t] \text{ EXCEPT } ![i].rollbackIndex = rollbackIndex,$
                                              $![i].rollbackValues = rollbackValues,$
                                              $![i].status = Validated]]$
                         $\land \text{UNCHANGED } \langle configuration \rangle$
                       $\lor \land r = Invalid$
                         $\land configuration' = [configuration \text{ EXCEPT } ![t].committedIndex = i]$
                         $\land proposal' = [proposal \text{ EXCEPT } ![t] = [$
                                       $proposal[t] \text{ EXCEPT } ![i].status = Failed]]$
              If the Rollback target is not the most recent change to the configuration,
              fail validation for the proposal.
             $\lor \land configuration[t].configIndex \neq proposal[t][i].rollback$
               $\land configuration' = [configuration \text{ EXCEPT } ![t].committedIndex = i]$
               $\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].status = Failed]]$
         If a Rollback proposal is attempting to roll back another Rollback,
         fail validation for the proposal.
         $\lor \land proposal[t][proposal[t][i].rollback].type = Rollback$
           $\land configuration' = [configuration \text{ EXCEPT } ![t].committedIndex = i]$
           $\land proposal' = [proposal \text{ EXCEPT } ![t] = [$
               $proposal[t] \text{ EXCEPT } ![i].status = Failed]]$
  $\land \text{UNCHANGED } \langle target \rangle$
 While in the Committing state, commit the proposed changes to the configuration.
$\lor \land proposal[t][i].status = Committing$
   Only commit the proposal if the prior proposal has already been committed.
  $\land configuration[t].committedIndex = proposal[t][i].dependencyIndex$
     If the proposal is a change, commit the change values and set the configuration
```

```
        index to the proposal index.
      ∧ ∨ ∧ proposal[t][i].type = Change
          ∧ configuration′ = [configuration EXCEPT ![t].values         = proposal[t][i].values,
                                                   ![t].configIndex    = i,
                                                   ![t].committedIndex = i]
        If the proposal is a rollback, commit the rollback values and index. This
        will cause the configuration index to be reverted to the index prior to
        the transaction/proposal being rolled back.
        ∨ ∧ proposal[t][i].type = Rollback
          ∧ configuration′ = [configuration EXCEPT ![t].values         = proposal[t][i].rollbackValues,
                                                   ![t].configIndex    = proposal[t][i].rollbackIndex,
                                                   ![t].committedIndex = i]
      ∧ proposal′ = [proposal EXCEPT ![t] = [proposal[t] EXCEPT ![i].status = Committed]]
      ∧ UNCHANGED ⟨target⟩
    While in the Applying state, apply the proposed changes to the target.
    ∨ ∧ proposal[t][i].status = Applying
      ∧ configuration[t].appliedIndex = proposal[t][i].dependencyIndex
      ∧ configuration[t].appliedTerm = mastership[t].term
      ∧ mastership[t].master = n
       Model successful and failed target update requests.
      ∧ ∃ r ∈ {Success, Failure} :
          ∨ ∧ r = Success
                  If the proposal is a change, apply the change values to the target
                  and update the configuration's applied index and values.
            ∧ ∨ ∧ proposal[t][i].type = Change
                ∧ target′ = [target EXCEPT ![t] = proposal[t][i].values @@ target[t]]
                ∧ configuration′ = [configuration EXCEPT
                    ![t].appliedIndex = i,
                    ![t].appliedValues = proposal[t][i].values @@ configuration[t].appliedValues]
                  If the proposal is a rollback, apply the rollback values and update the
                  configuration's applied values with the rolled back values.
              ∨ ∧ proposal[t][i].type = Rollback
                ∧ target′ = [target EXCEPT ![t] = proposal[t][i].rollbackValues @@ target[t]]
                ∧ configuration′ = [configuration EXCEPT
                    ![t].appliedIndex  = i,
                    ![t].appliedValues = proposal[t][i].rollbackValues @@ configuration[t].appliedValues]
            ∧ proposal′ = [proposal EXCEPT ![t] = [proposal[t] EXCEPT ![i].status = Applied]]
            If the proposal could not be applied, update the configuration's applied index
            and mark the proposal Failed.
          ∨ ∧ r = Failure
            ∧ configuration′ = [configuration EXCEPT ![t].appliedIndex = i]
            ∧ proposal′ = [proposal EXCEPT ![t] = [proposal[t] EXCEPT ![i].status = Failed]]
            ∧ UNCHANGED ⟨target⟩
∧ UNCHANGED ⟨transaction, mastership⟩
```

This section models the Configuration reconciler.

```
ReconcileConfiguration(n, t) ≜
    ∧ ∨ ∧ Target[t].persistent
        ∧ configuration[t].status ≠ Persisted
        ∧ configuration′ = [configuration EXCEPT ![t].status = Persisted]
        ∧ UNCHANGED ⟨target⟩
      ∨ ∧ ¬Target[t].persistent
        ∧ mastership[t].term > configuration[t].configTerm
        ∧ configuration′ = [configuration EXCEPT ![t].configTerm = mastership[t].term,
                                                 ![t].status     = Synchronizing]
        ∧ UNCHANGED ⟨target⟩
      ∨ ∧ ¬Target[t].persistent
        ∧ configuration[t].status ≠ Pending
        ∧ mastership[t].term = configuration[t].configTerm
        ∧ mastership[t].master = Nil
        ∧ configuration′ = [configuration EXCEPT ![t].status = Pending]
        ∧ UNCHANGED ⟨target⟩
      ∨ ∧ configuration[t].status = Synchronizing
        ∧ mastership[t].term = configuration[t].configTerm
        ∧ mastership[t].master = n
        ∧ target′ = [target EXCEPT ![t] = configuration[t].appliedValues]
        ∧ configuration′ = [configuration EXCEPT ![t].appliedTerm = mastership[t].term,
                                                 ![t].status      = Synchronized]
    ∧ UNCHANGED ⟨proposal, transaction, mastership⟩
```

Init and next state predicates

```
Init ≜
    ∧ transaction = ⟨⟩
    ∧ proposal = [t ∈ DOMAIN Target ↦
                    [p ∈ {} ↦ [status     ↦ Initializing]]]
    ∧ configuration = [t ∈ DOMAIN Target ↦
                         [target ↦ t,
                          status ↦ Pending,
                          values ↦
                             [path ∈ {}  ↦
                                 [path    ↦ path,
                                  value   ↦ Nil,
                                  index   ↦ 0,
                                  deleted ↦ FALSE]],
                          configIndex     ↦ 0,
                          configTerm      ↦ 0,
                          $proposedIndex \mapsto 0,$
                          $committedIndex \mapsto 0,$
                          $appliedIndex \mapsto 0,$
                          $appliedTerm \mapsto 0,$
                          $appliedValues \mapsto$
                            $[path \in \{\} \mapsto$
                               $[path \mapsto path,$
                                $value \mapsto Nil,$
                                $index \mapsto 0,$
                                $deleted \mapsto \text{FALSE}]]]]$
$\land target = [t \in \text{DOMAIN } Target \mapsto$
               $[path \in \{\} \mapsto$
                  $[value \mapsto Nil]]]$
$\land mastership = [t \in \text{DOMAIN } Target \mapsto [master \mapsto Nil, term \mapsto 0]]$

$Next \triangleq$
$\lor \exists c \in ValidChanges :$
    $RequestChange(c)$
$\lor \exists t \in \text{DOMAIN } transaction :$
    $RequestRollback(t)$
$\lor \exists n \in Node :$
    $\exists t \in \text{DOMAIN } Target :$
      $SetMaster(n, t)$
$\lor \exists t \in \text{DOMAIN } Target :$
    $UnsetMaster(t)$
$\lor \exists n \in Node :$
    $\exists t \in \text{DOMAIN } transaction :$
        $ReconcileTransaction(n, t)$
$\lor \exists n \in Node :$
    $\exists t \in \text{DOMAIN } proposal :$
      $\exists i \in \text{DOMAIN } proposal[t] :$
          $ReconcileProposal(n, t, i)$
$\lor \exists n \in Node :$
    $\exists c \in \text{DOMAIN } configuration :$
        $ReconcileConfiguration(n, c)$

$Spec \triangleq Init \land \Box[Next]_{vars}$

$Order \triangleq$
$\land \forall i, j \in \text{DOMAIN } transaction :$
      $\lor j \leq i$
      $\lor Phase(transaction[i].status) \geq Phase(transaction[j].status)$
      $\lor transaction[j].status = Failed$
$\land \forall t \in \text{DOMAIN } proposal :$
      $\forall i, j \in \text{DOMAIN } proposal[t] :$
        $\lor j \leq i$
      $\lor$ $Phase(proposal[t][i].status) \geq Phase(proposal[t][j].status)$
      $\lor$ $proposal[t][i].status = Failed$

$Consistency \triangleq$
$\forall t \in \text{DOMAIN } target :$
  LET
      Compute the transaction indexes that have been applied to the target
      $appliedIndexes \triangleq \{i \in \text{DOMAIN } transaction :$
                        $\land transaction[i].type = Change$
                        $\land i \in \text{DOMAIN } proposal[t]$
                        $\land proposal[t][i].status = Applied$
                        $\land t \in \text{DOMAIN } transaction[i].values$
                        $\land \neg \exists j \in \text{DOMAIN } transaction :$
                              $\land j > i$
                              $\land transaction[j].type = Rollback$
                              $\land transaction[j].rollback = i$
                              $\land transaction[j].status = Applied\}$
      Compute the set of paths in the target that have been updated by transactions
      $appliedPaths \triangleq \text{UNION } \{\text{DOMAIN } transaction[i].values[t] : i \in appliedIndexes\}$
      Compute the highest index applied to the target for each path
      $pathIndexes \triangleq [p \in appliedPaths \mapsto \text{CHOOSE } i \in appliedIndexes :$
                            $\forall j \in appliedIndexes :$
                              $\land i \geq j$
                              $\land p \in \text{DOMAIN } transaction[i].values]$
      Compute the expected target configuration based on the last indexes applied
      to the target for each path.
      $expectedConfig \triangleq [p \in \text{DOMAIN } pathIndexes \mapsto transaction[pathIndexes[p]].values[p]]$
  IN
      $target[t] = expectedConfig$

$Isolation \triangleq$
$\forall i, j \in \text{DOMAIN } transaction :$
    $\lor j \leq i$
    $\lor transaction[i].targets \cap transaction[j].targets = \{\}$
    $\lor transaction[i].isolation \neq Serializable$
    $\lor$ $\land$ $\lor Phase(transaction[i].status) \geq Phase(Committed)$
           $\lor Phase(transaction[j].status) < Phase(Committing)$
       $\land$ $\lor Phase(transaction[i].status) \geq Phase(Applied)$
           $\lor Phase(transaction[j].status) < Phase(Applying)$
    $\lor transaction[j].status = Failed$

THEOREM $Safety \triangleq Spec \Rightarrow \Box(Order \land Consistency \land Isolation)$

$Completion \triangleq$
$\land \forall i \in \text{DOMAIN } transaction :$
    $\land transaction[i].status \in \{Committed, Failed\}$
```

```
          ∧ transaction[i].status ∈ {Applied, Failed}
    ∧ ∀ t ∈ DOMAIN proposal :
         ∀ i ∈ DOMAIN proposal[t] :
            ∧ proposal[t][i].status ∈ {Committed, Failed}
            ∧ proposal[t][i].status ∈ {Applied, Failed}

THEOREM Liveness ≜ Spec ⇒ ◇Completion
```

* Modification History

* Last modified *Mon Feb* 07 00:14:47 *PST* 2022 by *jordanhalterman*

* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*