```
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

Phase constants
CONSTANTS
    Initialize,
    Validate,
    Abort,
    Commit,
    Apply

Phase ≜
    {Initialize,
     Validate,
     Abort,
     Commit,
     Apply}

Status constants
CONSTANTS
    InProgress,
    Complete,
    Failed

State ≜
    {InProgress,
     Complete,
    Failed}
```

State constants

```
CONSTANTS
    Pending,
    Validated,
    Committed,
    Applied,
    Aborted

Status ≜
    {Pending,
     Validated,
     Committed,
     Applied,
     Aborted}

CONSTANTS
    Valid,
    Invalid

CONSTANTS
    Success,
    Failure
```

The set of all nodes

```
CONSTANT Node
```

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

```
Example: Target ≜ [
    target1 ↦ [persistent ↦ FALSE, values ↦ [
        path1 ↦ {"value1", "value2"},
        path2 ↦ {"value2", "value3"}]],
    target2 ↦ [persistent ↦ TRUE, values ↦ [
        path2 ↦ {"value3", "value4"},
        path3 ↦ {"value4", "value5"}]]]
```

```
CONSTANT Target
```

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 Phase ::= phase ∈
```

```
  {Initialize,
   Validate,
   Abort,
   Commit,
   Apply}

TYPE State ::= state ∈
  {InProgress,
   Complete,
   Failed}

TYPE Status ::= status ∈
  {Pending,
   Validated,
   Committed,
   Applied,
   Aborted}

TYPE Isolation ::= isolation ∈
  {ReadCommitted,
   Serializable}

TYPE Transaction ≜ [
  type      ::= type ∈ Type,
  index     ::= index ∈ Nat,
  isolation ::= isolation ∈ Isolation
  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)
  phase     ::= phase ∈ Phase,
  state     ::= state ∈ State,
  status ::= status ∈ Status]

TYPE Proposal ≜ [
  type    ::= type ∈ Type,
  txIndex ::= txIndex ∈ Nat,
  values ::= [ path ∈ SUBSET (DOMAIN Target[target].values) ↦ [
      value ::= value ∈ STRING,
      delete ::= delete ∈ BOOLEAN ]],
  rollback ::= index ∈ Nat,
  depIndex ::= depIndex ∈ Nat,
  rbIndex ::= rbIndex ∈ Nat,
  rbValues ::= [ path ∈ SUBSET (DOMAIN Target[target].values) ↦ [
      value ::= value ∈ STRING,
      delete ::= delete ∈ BOOLEAN ]],
  phase ::= phase ∈ Phase,
  state ::= state ∈ State]

TYPE Configuration ≜ [
  id       ::= id ∈ STRING,
  target ::= target ∈ STRING,
  values ::=  [ path ∈ SUBSET  (DOMAIN Target[target])  ↦  [
      value ::= value ∈ STRING,
      index ::= index ∈ Nat,
      deleted ::= delete ∈ BOOLEAN ]],
  cfgIndex ::= cfgIndex ∈ Nat,
  cgfTerm ::= cgfTerm ∈ Nat,
  txIndex ::= txIndex ∈ Nat,
  cmtIndex ::= cmtIndex ∈ Nat,
  tgtIndex ::= tgtIndex ∈ Nat,
  tgtTerm ::= tgtTerm ∈ Nat,
  tgtValues ::=  [ path ∈ SUBSET  (DOMAIN Target[target])  ↦  [
      value ::= value ∈ STRING,
      index ::= index ∈ Nat,
      deleted ::= delete ∈ BOOLEAN ]],
  state     ::= state ∈ State]
```

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 ≜ ⟨transaction, proposal, configuration, mastership, target⟩
```

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

```
SetMaster(n, t) ≜
    ∧ mastership[t].master ≠ n
    ∧ mastership′ = [mastership EXCEPT ![t].term   = mastership[t].term + 1,
                                        ![t].master = n]
    ∧ UNCHANGED ⟨transaction, proposal, configuration, target⟩

UnsetMaster(t) ≜
```

```
$\land\ mastership[t].master \neq Nil$
$\land\ mastership' = [mastership$ EXCEPT $![t].master = Nil]$
$\land$ UNCHANGED $\langle transaction, proposal, configuration, target\rangle$
```

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

```
$Value(s, t, p) \triangleq$
    LET $value \triangleq$ CHOOSE $v \in s : v.target = t \land v.path = p$
    IN
        $[value \mapsto value.value,$
        $\ delete \mapsto value.delete]$

$Paths(s, t) \triangleq$
    $[p \in \{v.path : v \in \{v \in s : v.target = t\}\} \mapsto Value(s, t, p)]$

$Changes(s) \triangleq$
    $[t \in \{v.target : v \in s\} \mapsto Paths(s, t)]$

$ValidValues(t, p) \triangleq$
    UNION $\{\{[value \mapsto v, delete \mapsto$ FALSE$] : v \in Target[t].values[p]\}, \{[value \mapsto Nil, delete \mapsto$ TRUE$]\}\}$

$ValidPaths(t) \triangleq$
    UNION $\{\{v$ @@ $[path \mapsto p] : v \in ValidValues(t, p)\} : p \in$ DOMAIN $Target[t].values\}$

$ValidTargets \triangleq$
    UNION $\{\{p$ @@ $[target \mapsto t] : p \in ValidPaths(t)\} : t \in$ DOMAIN $Target\}$
```

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.

```
$ValidChanges \triangleq$
    LET $changeSets \triangleq \{s \in$ SUBSET $ValidTargets :$
                        $\forall t \in$ DOMAIN $Target\ :$
                          $\forall p \in$ DOMAIN $Target[t].values :$
                            $Cardinality(\{v \in s : v.target = t \land v.path = p\}) \leq 1\}$
    IN
        $\{Changes(s) : s \in changeSets\}$
```

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.

```
$NextIndex \triangleq$
    IF DOMAIN $transaction = \{\}$ THEN
        $1$
     ELSE
        LET $i \triangleq$ CHOOSE $i \in$ DOMAIN $transaction :$
               $\forall j \in$ DOMAIN $transaction : i \geq j$
    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   ↦ {},
                                                       phase     ↦ Initialize,
                                                       state     ↦ InProgress,
                                                       status    ↦ Pending])
   ∧ 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   ↦ {},
                                                       phase     ↦ Initialize,
                                                       state     ↦ InProgress,
                                                       status    ↦ Pending])
   ∧ 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) ≜
```

Initialize 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.
$\land \lor \land transaction[i].phase = Initialize$
      $\land \lor \land transaction[i].state = InProgress$
            Serialize transaction initialization
            $\land \neg \exists j \in \textsc{domain}\ transaction :$
                  $\land j < i$
                  $\land transaction[j].phase = Initialize$
                  $\land transaction[j].state\ \ = InProgress$
            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.
                  $\land \lor \land transaction[i].type = Change$
                           $\land transaction' = [transaction\ \textsc{except}\ ![i].targets = \textsc{domain}\ transaction[i].values]$
                           $\land proposal' = [t \in \textsc{domain}\ proposal \mapsto$
                                $\textsc{if}\ \ t \in \textsc{domain}\ transaction[i].values\ \textsc{then}$
                                   $proposal[t] @@ (i :> [type \mapsto Change,$
                                                          $index \mapsto i,$
                                                          $values \mapsto transaction[i].values[t],$
                                                          $depIndex \mapsto 0,$
                                                          $rbIndex \mapsto 0,$
                                                          $rbValues \mapsto \langle\rangle,$
                                                          $phase \mapsto Initialize,$
                                                          $state \mapsto InProgress])$
                                 $\textsc{else}$
                                   $proposal[t]]$
                     If the transaction is a rollback, the targets affected are
                     the targets of the change transaction being rolled back.
                     $\lor \land transaction[i].type = Rollback$
                        $\land \lor \land transaction[i].rollback \in \textsc{domain}\ transaction$
                              $\land transaction[transaction[i].rollback].type = Change$
                              $\land transaction' = [transaction\ \textsc{except}\ ![i].targets =$
                                                    $\textsc{domain}\ transaction[transaction[i].rollback].values]$
                              $\land proposal' = [t \in \textsc{domain}\ proposal \mapsto$
                                   $\textsc{if}\ \ t \in \textsc{domain}\ transaction[transaction[i].rollback].values\ \textsc{then}$
                                      $proposal[t] @@ (i :> [type \mapsto Rollback,$
                                                             $index \mapsto i,$
                                                             $rollback \mapsto transaction[i].rollback,$
                                                             $depIndex \mapsto 0,$
                                                             $rbIndex \mapsto 0,$
                                                             $rbValues \mapsto \langle\rangle,$
                                                             $phase \mapsto Initialize,$
                                                             $state \mapsto InProgress])$
                                    $\textsc{else}$
                        $proposal[t]]$
                $\lor\ \land\ \lor\ \land\ transaction[i].rollback \in \text{DOMAIN}\ transaction$
                      $\land\ transaction[transaction[i].rollback].type = Rollback$
                    $\lor\ transaction[i].rollback \notin \text{DOMAIN}\ transaction$
                  $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state = Failed]$
                  $\land\ \text{UNCHANGED}\ \langle proposal \rangle$
        $\lor\ \land\ transaction[i].targets \neq \{\}$
             If all proposals have been Complete, mark the transaction Complete.
          $\land\ \lor\ \land \forall\, t \in transaction[i].targets :$
                  $\land\ proposal[t][i].phase\ \ = Initialize$
                  $\land\ proposal[t][i].state = Complete$
              $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state = Complete]$
              $\land\ \text{UNCHANGED}\ \langle proposal \rangle$
             If any proposal has been Failed, mark the transaction Failed.
            $\lor\ \land \exists\, t \in transaction[i].targets :$
                  $\land\ proposal[t][i].phase\ \ = Initialize$
                  $\land\ proposal[t][i].state = Failed$
              $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state = Failed]$
              $\land\ \text{UNCHANGED}\ \langle proposal \rangle$
    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.
    $\lor\ \land\ transaction[i].state = Complete$
      $\land \forall\, t \in transaction[i].targets :$
         $\land\ proposal[t][i].depIndex \in transaction$
         $\land\ transaction[proposal[t][i].depIndex].isolation = Serializable$
         $\Rightarrow transaction[proposal[t][i].depIndex].status \in \{Validated, Committed, Applied, Aborted\}$
      $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].phase = Validate,$
                                     $![i].state\ \ = InProgress]$
      $\land\ \text{UNCHANGED}\ \langle proposal \rangle$
    $\lor\ \land\ transaction[i].state = Failed$
      $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].phase = Abort,$
                                     $![i].state\ \ = InProgress]$
      $\land\ \text{UNCHANGED}\ \langle proposal \rangle$
$\lor\ \land\ transaction[i].phase = Validate$
  $\land\ \lor\ \land\ transaction[i].state = InProgress$
         Move the transaction's proposals to the Validating state
      $\land\ \lor\ \land \exists\, t \in transaction[i].targets :$
             $\land\ proposal[t][i].phase \neq Validate$
             $\land\ proposal' = [proposal\ \text{EXCEPT}\ ![t] = [$
                           $proposal[t]\ \text{EXCEPT}\ ![i].phase = Validate,$
                                            $![i].state\ \ = InProgress]]$
          $\land\ \text{UNCHANGED}\ \langle transaction \rangle$
         If all proposals have been Complete, mark the transaction Complete.
```

```
          ∨ ∧ ∀ t ∈ transaction[i].targets :
                  ∧ proposal[t][i].phase  = Validate
                  ∧ proposal[t][i].state = Complete
            ∧ transaction′ = [transaction EXCEPT ![i].state  = Complete,
                                                 ![i].status = Validated]
            ∧ UNCHANGED ⟨proposal⟩
          If any proposal has been Failed, mark the transaction Failed.
          ∨ ∧ ∃ t ∈ transaction[i].targets :
                  ∧ proposal[t][i].phase  = Validate
                  ∧ proposal[t][i].state = Failed
            ∧ transaction′ = [transaction EXCEPT ![i].state = Failed]
            ∧ UNCHANGED ⟨proposal⟩
      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.
      ∨ ∧ transaction[i].state = Complete
        ∧ ∀ t ∈ transaction[i].targets :
            ∧ proposal[t][i].depIndex ∈ transaction
            ∧ transaction[proposal[t][i].depIndex].isolation = Serializable
            ⇒ transaction[proposal[t][i].depIndex].status ∈ {Committed, Applied, Aborted}
        ∧ transaction′ = [transaction EXCEPT ![i].phase = Commit,
                                             ![i].state  = InProgress]
        ∧ UNCHANGED ⟨proposal⟩
      ∨ ∧ transaction[i].state = Failed
        ∧ transaction′ = [transaction EXCEPT ![i].phase = Abort,
                                             ![i].state  = InProgress]
        ∧ UNCHANGED ⟨proposal⟩
  ∨ ∧ transaction[i].phase = Commit
    ∧ ∨ ∧ transaction[i].state = InProgress
          Move the transaction’s proposals to the Committing state
        ∧ ∨ ∧ ∃ t ∈ transaction[i].targets :
                  ∧ proposal[t][i].phase ≠ Validate
                  ∧ proposal′ = [proposal EXCEPT ![t] = [
                                   proposal[t] EXCEPT ![i].phase = Commit,
                                                      ![i].state  = InProgress]]
            ∧ UNCHANGED ⟨transaction⟩
          If all proposals have been Complete, mark the transaction Complete.
          ∨ ∧ ∀ t ∈ transaction[i].targets :
                  ∧ proposal[t][i].phase  = Commit
                  ∧ proposal[t][i].state = Complete
            ∧ transaction′ = [transaction EXCEPT ![i].state  = Complete,
                                                 ![i].status = Committed]
            ∧ UNCHANGED ⟨proposal⟩
      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.
          $\lor \land transaction[i].state = Complete$
            $\land \forall t \in transaction[i].targets :$
                $\land proposal[t][i].depIndex \in transaction$
                $\land transaction[proposal[t][i].depIndex].isolation = Serializable$
                $\Rightarrow transaction[proposal[t][i].depIndex].status \in \{Applied, Aborted\}$
            $\land transaction' = [transaction \text{ EXCEPT } ![i].phase = Apply,$
                                                  $![i].state = InProgress]$
            $\land \text{UNCHANGED } \langle proposal \rangle$
      $\lor \land transaction[i].phase = Apply$
        $\land transaction[i].state = InProgress$
          Move the transaction's proposals to the Applying state
        $\land \lor \land \exists t \in transaction[i].targets :$
                $\land proposal[t][i].phase \neq Validate$
                $\land proposal' = [proposal \text{ EXCEPT } ![t] = [$
                          $proposal[t] \text{ EXCEPT } ![i].phase = Apply,$
                                                  $![i].state = InProgress]]$
            $\land \text{UNCHANGED } \langle transaction \rangle$
          If all proposals have been Complete, mark the transaction Complete.
          $\lor \land \forall t \in transaction[i].targets :$
                $\land proposal[t][i].phase = Apply$
                $\land proposal[t][i].state = Complete$
            $\land transaction' = [transaction \text{ EXCEPT } ![i].state = Complete,$
                                                  $![i].status = Applied]$
            $\land \text{UNCHANGED } \langle proposal \rangle$
          If any proposal has been Failed, mark the transaction Failed.
          $\lor \land \exists t \in transaction[i].targets :$
                $\land proposal[t][i].phase = Apply$
                $\land proposal[t][i].state = Failed$
            $\land transaction' = [transaction \text{ EXCEPT } ![i].state = Failed]$
            $\land \text{UNCHANGED } \langle proposal \rangle$
      The Aborting state is used to clean up transactions that have failed during
      the Initializing or Validating phases.
      $\lor \land transaction[i].phase = Abort$
        $\land transaction[i].state = InProgress$
          Move the transaction's proposals to the Aborting state
        $\land \lor \land \exists t \in transaction[i].targets :$
                $\land proposal[t][i].phase \neq Validate$
                $\land proposal' = [proposal \text{ EXCEPT } ![t] = [$
                          $proposal[t] \text{ EXCEPT } ![i].phase = Abort,$
                                                  $![i].state = InProgress]]$
            $\land \text{UNCHANGED } \langle transaction \rangle$
          If all proposals have been Complete, mark the transaction Complete.
        $\lor\ \land\ \forall\, t \in \mathit{transaction}[i].\mathit{targets} :$
                  $\land\ \mathit{proposal}[t][i].\mathit{phase}\ = \mathit{Abort}$
                  $\land\ \mathit{proposal}[t][i].\mathit{state} = \mathit{Complete}$
           $\land\ \mathit{transaction}' = [\mathit{transaction}\ \text{EXCEPT}\ ![i].\mathit{state}\ \ = \mathit{Complete},$
                                                          $![i].\mathit{status} = \mathit{Aborted}]$
           $\land\ \text{UNCHANGED}\ \langle \mathit{proposal} \rangle$
  $\land\ \text{UNCHANGED}\ \langle \mathit{configuration},\ \mathit{mastership},\ \mathit{target} \rangle$

Reconcile a proposal
$\mathit{ReconcileProposal}(n,\ t,\ i)\ \triangleq$
  $\land\ \lor\ \land\ \mathit{proposal}[t][i].\mathit{phase} = \mathit{Initialize}$
        $\land\ \mathit{proposal}[t][i].\mathit{state}\ = \mathit{InProgress}$
        $\land\ \mathit{proposal}' = [\mathit{proposal}\ \text{EXCEPT}\ ![t] = [$
            $\mathit{proposal}[t]\ \text{EXCEPT}\ ![i] = [$
              $\mathit{state}\qquad \mapsto \mathit{Complete},$
              $\mathit{depIndex} \mapsto \mathit{configuration}[t].\mathit{txIndex}]\ @@\ \mathit{proposal}[t][i]]]$
        $\land\ \mathit{configuration}' = [\mathit{configuration}\ \text{EXCEPT}\ ![t].\mathit{txIndex} = i]$
        $\land\ \text{UNCHANGED}\ \langle \mathit{target} \rangle$
      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.
     $\lor\ \land\ \mathit{proposal}[t][i].\mathit{phase} = \mathit{Validate}$
        $\land\ \mathit{proposal}[t][i].\mathit{state}\ = \mathit{InProgress}$
        $\land\ \mathit{configuration}[t].\mathit{cmtIndex} = \mathit{proposal}[t][i].\mathit{depIndex}$
          For Change proposals validate the set of requested changes.
        $\land\ \lor\ \land\ \mathit{proposal}[t][i].\mathit{type} = \mathit{Change}$
              $\land\ \text{LET}\ \mathit{rbIndex}\ \triangleq\ \mathit{configuration}[t].\mathit{cfgIndex}$
                    $\mathit{rbValues}\ \triangleq\ [p \in \text{DOMAIN}\ \mathit{proposal}[t][i].\mathit{values} \mapsto$
                                $\text{IF}\ p \in \text{DOMAIN}\ \mathit{configuration}[t].\mathit{values}\ \text{THEN}$
                                  $\mathit{configuration}[t].\mathit{values}[p]$
                                 $\text{ELSE}$
                                  $[\mathit{value}\ \ \mapsto \mathit{Nil},$
                                   $\mathit{delete} \mapsto \text{TRUE}]]$
                Model validation successes and failures with Valid and Invalid results.
                $\text{IN}\quad \exists\, r \in \{\mathit{Valid},\ \mathit{Invalid}\} :$
                      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.
                     $\lor\ \land\ r = \mathit{Valid}$
                        $\land\ \mathit{proposal}' = [\mathit{proposal}\ \text{EXCEPT}\ ![t] = [$
                                      $\mathit{proposal}[t]\ \text{EXCEPT}\ ![i].\mathit{rbIndex}\ \ = \mathit{rbIndex},$
                                                          $![i].\mathit{rbValues} = \mathit{rbValues},$
                                                          $![i].\mathit{state}\qquad = \mathit{Complete}]]$
                     $\lor\ \land\ r = \mathit{Invalid}$
                        $\land\ \mathit{proposal}' = [\mathit{proposal}\ \text{EXCEPT}\ ![t] = [$
```

```
                            $proposal[t]\ \text{EXCEPT}\ ![i].state = Failed]]$
     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.
          $\land \lor \land configuration[t].cfgIndex = proposal[t][i].rollback$
              $\land \text{LET}\ rbIndex \triangleq proposal[t][proposal[t][i].rollback].rbIndex$
                    $rbValues \triangleq proposal[t][proposal[t][i].rollback].rbValues$
                $\text{IN}\ \exists r \in \{Valid, Invalid\} :$
                      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.
                     $\lor \land r = Valid$
                       $\land proposal' = [proposal\ \text{EXCEPT}\ ![t] = [$
                           $proposal[t]\ \text{EXCEPT}\ ![i].rbIndex = rbIndex,$
                                           $![i].rbValues = rbValues,$
                                           $![i].state = Complete]]$
                     $\lor \land r = Invalid$
                       $\land proposal' = [proposal\ \text{EXCEPT}\ ![t] = [$
                                     $proposal[t]\ \text{EXCEPT}\ ![i].state = Failed]]$
             If the Rollback target is not the most recent change to the configuration,
             fail validation for the proposal.
            $\lor \land configuration[t].cfgIndex \neq proposal[t][i].rollback$
              $\land proposal' = [proposal\ \text{EXCEPT}\ ![t] = [proposal[t]\ \text{EXCEPT}\ ![i].state = Failed]]$
         If a Rollback proposal is attempting to roll back another Rollback,
         fail validation for the proposal.
        $\lor \land proposal[t][proposal[t][i].rollback].type = Rollback$
          $\land proposal' = [proposal\ \text{EXCEPT}\ ![t] = [$
              $proposal[t]\ \text{EXCEPT}\ ![i].state = Failed]]$
  $\land \text{UNCHANGED}\ \langle configuration, target\rangle$
While in the Commit state, commit the proposed changes to the configuration.
$\lor \land proposal[t][i].phase = Commit$
  $\land proposal[t][i].state = InProgress$
  Only commit the proposal if the prior proposal has already been committed.
  $\land configuration[t].cmtIndex = proposal[t][i].depIndex$
     If the proposal is a change, commit the change values and set the configuration
     index to the proposal index.
  $\land \lor \land proposal[t][i].type = Change$
      $\land configuration' = [configuration\ \text{EXCEPT}\ ![t].values = proposal[t][i].values,$
                                        $![t].cfgIndex = i,$
                                        $![t].cmtIndex = i]$
     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.
          $\lor \land proposal[t][i].type = Rollback$
            $\land configuration' = [configuration \text{ EXCEPT } ![t].values \quad = proposal[t][i].rbValues,$
                                                 $![t].cfgIndex \quad = proposal[t][i].rbIndex,$
                                                 $![t].cmtIndex = i]$
      $\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = Complete]]$
      $\land \text{UNCHANGED } \langle target \rangle$
    While in the Apply phase, apply the proposed changes to the target.
    $\lor \land proposal[t][i].phase = Apply$
      $\land proposal[t][i].state \quad = InProgress$
      $\land configuration[t].tgtIndex = proposal[t][i].depIndex$
      $\land configuration[t].tgtTerm = mastership[t].term$
      $\land mastership[t].master = n$
       Model successful and failed target update requests.
      $\land \exists\, r \in \{Success, Failure\} :$
          $\lor \land r = Success$
                If the proposal is a change, apply the change values to the target
                and update the configuration's applied index and values.
             $\land \lor \land proposal[t][i].type = Change$
                    $\land target' = [target \text{ EXCEPT } ![t] = proposal[t][i].values \mathbin{@@} target[t]]$
                    $\land configuration' = [configuration \text{ EXCEPT }$
                          $![t].tgtIndex = i,$
                          $![t].tgtValues = proposal[t][i].values \mathbin{@@} configuration[t].tgtValues]$
                If the proposal is a rollback, apply the rollback values and update the
                configuration's applied values with the rolled back values.
                $\lor \land proposal[t][i].type = Rollback$
                    $\land target' = [target \text{ EXCEPT } ![t] = proposal[t][i].rbValues \mathbin{@@} target[t]]$
                    $\land configuration' = [configuration \text{ EXCEPT }$
                          $![t].tgtIndex \quad = i,$
                          $![t].tgtValues = proposal[t][i].rbValues \mathbin{@@} configuration[t].tgtValues]$
             $\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = Complete]]$
            If the proposal could not be applied, update the configuration's applied index
            and mark the proposal Failed.
          $\lor \land r = Failure$
             $\land configuration' = [configuration \text{ EXCEPT } ![t].tgtIndex = i]$
             $\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = Failed]]$
             $\land \text{UNCHANGED } \langle target \rangle$
    $\lor \land proposal[t][i].phase = Abort$
      $\land proposal[t][i].state \quad = InProgress$
          The cmtIndex will always be greater than or equal to the tgtIndex.
          If only the cmtIndex matches the proposal's depIndex, update
          the cmtIndex to enable commits of later proposals, but do not
          mark the Abort phase Complete until the tgtIndex has been incremented.
      $\land \lor \land configuration[t].cmtIndex = proposal[t][i].depIndex$
```

```
            $\land$ $configuration' = [configuration$ EXCEPT $![t].cmtIndex = i]$
            $\land$ UNCHANGED $\langle proposal \rangle$
          If the configuration's tgtIndex matches the proposal's depIndex,
          update the tgtIndex and mark the proposal Complete for the Abort phase.
          $\lor$ $\land$ $configuration[t].cmtIndex \geq i$
            $\land$ $configuration[t].tgtIndex = proposal[t][i].depIndex$
            $\land$ $configuration' = [configuration$ EXCEPT $![t].tgtIndex = i]$
            $\land$ $proposal' = [proposal$ EXCEPT $![t] = [proposal[t]$ EXCEPT $![i].state = Complete]]$
          If both the configuration's cmtIndex and tgtIndex match the
          proposal's depIndex, update the cmtIndex and tgtIndex
          and mark the proposal Complete for the Abort phase.
          $\lor$ $\land$ $configuration[t].cmtIndex = proposal[t][i].depIndex$
            $\land$ $configuration[t].tgtIndex = proposal[t][i].depIndex$
            $\land$ $configuration' = [configuration$ EXCEPT $![t].cmtIndex = i,$
                                                  $![t].tgtIndex = i]$
            $\land$ $proposal' = [proposal$ EXCEPT $![t] = [proposal[t]$ EXCEPT $![i].state = Complete]]$
        $\land$ UNCHANGED $\langle target \rangle$
    $\land$ UNCHANGED $\langle transaction, mastership \rangle$
```

```
This section models the Configuration reconciler.
$ReconcileConfiguration(n, t) \triangleq$
    $\land$ $\lor$ $\land$ $Target[t].persistent$
          $\land$ $configuration[t].state \neq Complete$
          $\land$ $configuration' = [configuration$ EXCEPT $![t].state = Complete]$
          $\land$ UNCHANGED $\langle target \rangle$
       $\lor$ $\land$ $\neg Target[t].persistent$
          $\land$ $\lor$ $mastership[t].term > configuration[t].cfgTerm$
             $\lor$ $\land$ $mastership[t].term = configuration[t].cfgTerm$
                $\land$ $mastership[t].master = Nil$
          $\land$ $configuration' = [configuration$ EXCEPT $![t].cfgTerm = mastership[t].term,$
                                                $![t].state = InProgress]$
          $\land$ UNCHANGED $\langle target \rangle$
       $\lor$ $\land$ $configuration[t].state = InProgress$
          $\land$ $mastership[t].term = configuration[t].cfgTerm$
          $\land$ $mastership[t].master = n$
          $\land$ $target' = [target$ EXCEPT $![t] = configuration[t].tgtValues]$
          $\land$ $configuration' = [configuration$ EXCEPT $![t].tgtTerm = mastership[t].term,$
                                                $![t].state = Complete]$
    $\land$ UNCHANGED $\langle proposal, transaction, mastership \rangle$
```

Formal specification, constraints, and theorems.

```
Init ≜
    ∧ transaction = ⟨⟩
    ∧ proposal = [t ∈ DOMAIN Target ↦
                      [p ∈ {} ↦ [phase ↦ Initialize,
                                 state  ↦ InProgress]]]
    ∧ configuration = [t ∈ DOMAIN Target ↦
                          [target  ↦ t,
                           state   ↦ InProgress,
                           values ↦
                              [path ∈ {}  ↦
                                  [path     ↦ path,
                                   value    ↦ Nil,
                                   index    ↦ 0,
                                   deleted ↦ FALSE]],
                           cfgIndex  ↦ 0,
                           cfgTerm   ↦ 0,
                           txIndex    ↦ 0,
                           cmtIndex ↦ 0,
                           tgtIndex  ↦ 0,
                           tgtTerm   ↦ 0,
                           tgtValues ↦
                              [path ∈ {}  ↦
                                  [path     ↦ path,
                                   value    ↦ Nil,
                                   index    ↦ 0,
                                   deleted ↦ FALSE]]]]
    ∧ target = [t ∈ DOMAIN Target ↦
                   [path ∈ {} ↦
                       [value ↦ Nil]]]
    ∧ mastership = [t ∈ DOMAIN Target ↦ [master ↦ Nil, term ↦ 0]]

Next ≜
    ∨ ∃ c ∈ ValidChanges :
        RequestChange(c)
    ∨ ∃ t ∈ DOMAIN transaction :
        RequestRollback(t)
    ∨ ∃ n ∈ Node :
        ∃ t ∈ DOMAIN Target :
          SetMaster(n, t)
    ∨ ∃ t ∈ DOMAIN Target :
        UnsetMaster(t)
    ∨ ∃ n ∈ Node :
        ∃ t ∈ DOMAIN transaction :
            ReconcileTransaction(n, t)
    ∨ ∃ n ∈ Node :
          ∃ t ∈ DOMAIN proposal :
            ∃ i ∈ DOMAIN proposal[t] :
                ReconcileProposal(n, t, i)
    ∨ ∃ n ∈ Node :
          ∃ c ∈ DOMAIN configuration :
              ReconcileConfiguration(n, c)

Spec ≜ Init ∧ □[Next]_vars

Order ≜
   ∀ t ∈ DOMAIN proposal :
     ∀ i ∈ DOMAIN proposal[t] :
        ∧ ∧ proposal[t][i].phase = Commit
          ∧ proposal[t][i].state  = InProgress
          ⇒ ¬∃ j ∈ DOMAIN proposal[t] :
                  ∧ j > i
                  ∧ proposal[t][j].phase = Commit
                  ∧ proposal[t][j].state  = Complete
        ∧ ∧ proposal[t][i].phase = Apply
          ∧ proposal[t][i].state  = InProgress
          ⇒ ¬∃ j ∈ DOMAIN proposal[t] :
                  ∧ j > i
                  ∧ proposal[t][j].phase = Apply
                  ∧ proposal[t][j].state  = Complete

Consistency ≜
   ∀ t ∈ DOMAIN target :
     LET
          Compute the transaction indexes that have been applied to the target
         tgtIndexes ≜ {i ∈ DOMAIN transaction :
                            ∧ transaction[i].type = Change
                            ∧ i ∈ DOMAIN proposal[t]
                            ∧ proposal[t][i].phase = Apply
                            ∧ proposal[t][i].state  = Complete
                            ∧ t ∈ DOMAIN transaction[i].values
                            ∧ ¬∃ j ∈ DOMAIN transaction :
                                   ∧ j > i
                                   ∧ transaction[j].type = Rollback
                                   ∧ transaction[j].rollback = i
                                   ∧ transaction[j].phase = Apply
                                   ∧ transaction[j].state  = Complete}
          Compute the set of paths in the target that have been updated by transactions
         appliedPaths    ≜ UNION {DOMAIN transaction[i].values[t] : i ∈ tgtIndexes}
          Compute the highest index applied to the target for each path
         pathIndexes     ≜ [p ∈ appliedPaths ↦ CHOOSE i ∈ tgtIndexes :
                                   ∀ j ∈ tgtIndexes :
                                      ∧ i ≥ j
                                      ∧ p ∈ DOMAIN transaction[i].values[t]]
                Compute the expected target configuration based on the last indexes applied
                to the target for each path.
               expectedConfig ≜ [p ∈ DOMAIN pathIndexes ↦ transaction[pathIndexes[p]].values[t][p]]
           IN
               target[t] = expectedConfig

Isolation ≜
   ∀ i ∈ DOMAIN transaction :
      ∧ ∧ transaction[i].phase = Commit
        ∧ transaction[i].isolation = Serializable
        ⇒ ¬∃ j ∈ DOMAIN transaction :
               ∧ j > i
               ∧ transaction[j].targets ∩ transaction[i].targets ≠ {}
               ∧ transaction[j].phase = Commit
      ∧ ∧ transaction[i].phase = Apply
        ∧ transaction[i].isolation = Serializable
        ⇒ ¬∃ j ∈ DOMAIN transaction :
               ∧ j > i
               ∧ transaction[j].targets ∩ transaction[i].targets ≠ {}
               ∧ transaction[j].phase = Apply

Safety ≜ □(Order ∧ Consistency ∧ Isolation)

THEOREM Spec ⇒ Safety

Completion ≜
   ∀ i ∈ DOMAIN transaction :
      ∧ transaction[i].phase ∈ {Apply, Abort}
      ∧ transaction[i].state  = Complete

Liveness ≜ ◇Completion

THEOREM Spec ⇒ Liveness
```

```
Type assumptions.
ASSUME Nil ∈ STRING

ASSUME ∀ phase ∈ Phase : phase ∈ STRING

ASSUME ∀ state  ∈ State  : state  ∈ STRING

ASSUME ∀ status ∈ Status : status ∈ STRING

ASSUME ∧ IsFiniteSet(Node)
       ∧ ∀ n ∈ Node :
```

```
                    $\land\ n \notin \textsc{domain}\ Target$
                    $\land\ n \in \textsc{string}$

$\textsc{assume}$   $\land\ \forall\, t \in \textsc{domain}\ Target :$
                    $\land\ t \notin Node$
                    $\land\ t \in \textsc{string}$
                    $\land\ Target[t].persistent \in \textsc{boolean}$
                    $\land\ \forall\, p \in \textsc{domain}\ Target[t].values :$
                        $IsFiniteSet(Target[t].values[p])$
```

* Modification History
* Last modified *Tue Feb* 08 02:18:39 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*