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

$$
\begin{array}{l}
Phase \triangleq \\
\quad \text{LET } phases \triangleq \langle Initialize, \\
\qquad\qquad\qquad\qquad Validate, \\
\qquad\qquad\qquad\qquad Abort, \\
\qquad\qquad\qquad\qquad Commit, \\
\qquad\qquad\qquad\qquad Apply\rangle \\
\quad \text{IN } \quad [p \in \{phases[i] : i \in \text{DOMAIN } phases\} \mapsto \\
\qquad\quad \text{CHOOSE } i \in \text{DOMAIN } phases : phases[i] = p]
\end{array}
$$

Status constants

CONSTANTS
Pending,
Complete,
Failed

$$Status \triangleq$$

LET $statuses \triangleq \langle Pending,$
$Complete,$
$Failed\rangle$
IN $[s \in \{statuses[i] : i \in \text{DOMAIN } statuses\} \mapsto$
CHOOSE $i \in \text{DOMAIN } statuses : statuses[i] = s]$

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* $\triangleq$ [
*target*1 $\mapsto$ [*persistent* $\mapsto$ FALSE, *values* $\mapsto$ [
*path*1 $\mapsto$ {"*value*1", "*value*2"},
*path*2 $\mapsto$ {"*value*2", "*value*3"}]],
*target*2 $\mapsto$ [*persistent* $\mapsto$ TRUE, *values* $\mapsto$ [
*path*2 $\mapsto$ {"*value*3", "*value*4"},
*path*3 $\mapsto$ {"*value*4", "*value*5"}]]]

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* $\in$
{*Change*,
Rollback}

TYPE *Phase* ::= *phase* $\in$
{*Initialize*,
Validate,
Abort,
Commit,
Apply}

TYPE *Status* ::= *status* $\in$
{*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)
  phase     ::= phase ∈ Phase]
  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 ]],
  phase       ::= phase ∈ Phase]
  status      ::= status ∈ Status]

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

$SetMaster(n, t) \triangleq$
- $\land\ mastership[t].master \neq n$
- $\land\ mastership' = [mastership \text{ EXCEPT } ![t].term = mastership[t].term + 1,$
  $![t].master = n]$
- $\land$ UNCHANGED $\langle transaction, proposal, configuration, target \rangle$

$UnsetMaster(t) \triangleq$
- $\land\ mastership[t].master \neq Nil$
- $\land\ mastership' = [mastership \text{ EXCEPT } ![t].master = Nil]$
- $\land$ 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  ↦ value.value,
         delete ↦ value.delete]

Paths(s, t) ≜
   [p ∈ {v.path : v ∈ {v ∈ s : v.target = t}} ↦ Value(s, t, p)]

Changes(s) ≜
   [t ∈ {v.target : v ∈ s} ↦ Paths(s, t)]

ValidValues(t, p) ≜
   UNION {{[value ↦ v, delete ↦ FALSE] : v ∈ Target[t].values[p]}, {[value ↦ Nil, delete ↦ TRUE]}}

ValidPaths(t) ≜
   UNION {{v @@ [path ↦ p] : v ∈ ValidValues(t, p)} : p ∈ DOMAIN Target[t].values}

ValidTargets ≜
   UNION {{p @@ [target ↦ t] : p ∈ ValidPaths(t)} : t ∈ 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 ≜
   LET changeSets ≜ {s ∈ SUBSET ValidTargets :
                       ∀ t ∈ DOMAIN Target   :
                         ∀ 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   ↦ {},
```

$$\begin{array}{l} \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad phase \quad \mapsto Initialize, \\ \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad status \quad \mapsto Pending]) \\ \land \text{UNCHANGED } \langle proposal, configuration, mastership, target \rangle \end{array}$$

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

$$\begin{array}{l} RequestRollback(t) \triangleq \\ \quad \land \exists\, isolation \in \{ReadCommitted, Serializable\} : \\ \quad\quad \land transaction' = transaction \,@@\, (NextIndex :> [type \mapsto Rollback, \\ \quad\quad\quad index \mapsto NextIndex, \\ \quad\quad\quad isolation \mapsto isolation, \\ \quad\quad\quad rollback \mapsto t, \\ \quad\quad\quad targets \mapsto \{\}, \\ \quad\quad\quad phase \mapsto Initialize, \\ \quad\quad\quad status \mapsto Pending]) \\ \quad \land \text{UNCHANGED } \langle proposal, configuration, mastership, target \rangle \end{array}$$

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

$ReconcileTransaction(n, i) \triangleq$

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.

$$\begin{array}{l} \land \lor \land transaction[i].phase = Initialize \\ \quad\quad \land \lor \land transaction[i].status = Pending \end{array}$$

Serialize transaction initialization

$$\begin{array}{l} \land i - 1 \in \text{DOMAIN } transaction \Rightarrow \\ \quad\quad \lor Phase[transaction[i-1].phase] > Phase[Initialize] \\ \quad\quad \lor transaction[i-1].status \neq Pending \end{array}$$

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.

$\land \lor \land transaction[i].type = Change$
$\land transaction' = [transaction \text{ EXCEPT } ![i].targets = \text{DOMAIN } transaction[i].values]$
$\land proposal' = [t \in \text{DOMAIN } proposal \mapsto$
IF $t \in \text{DOMAIN } transaction[i].values$ THEN
$proposal[t] \mathbin{@@} (i :> [type \mapsto Change,$
$index \mapsto i,$
$values \mapsto transaction[i].values[t],$
$dependencyIndex \mapsto 0,$
$rollbackIndex \mapsto 0,$
$rollbackValues \mapsto \langle\rangle,$
$phase \mapsto Initialize,$
$status \mapsto Pending])$
ELSE
$proposal[t]]$

If the transaction is a rollback, the targets affected are the targets of the change transaction being rolled back.

$\lor \land transaction[i].type = Rollback$
$\land \lor \land transaction[i].rollback \in \text{DOMAIN } transaction$
$\land transaction[transaction[i].rollback].type = Change$
$\land transaction' = [transaction \text{ EXCEPT } ![i].targets =$
$\text{DOMAIN } transaction[transaction[i].rollback].values]$
$\land proposal' = [t \in \text{DOMAIN } proposal \mapsto$
IF $t \in \text{DOMAIN } transaction[transaction[i].rollback].values$ THEN
$proposal[t] \mathbin{@@} (i :> [type \mapsto Rollback,$
$index \mapsto i,$
$rollback \mapsto transaction[i].rollback,$
$dependencyIndex \mapsto 0,$
$rollbackIndex \mapsto 0,$
$rollbackValues \mapsto \langle\rangle,$
$phase \mapsto Initialize,$
$status \mapsto Pending])$
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 *Complete*, mark the transaction *Complete*.

$\land \lor \land \forall t \in transaction[i].targets :$
$\land proposal[t][i].phase = Initialize$
$\land proposal[t][i].status = Complete$
$\land transaction' = [transaction \text{ EXCEPT } ![i].status = Complete]$
$\land \text{UNCHANGED } \langle proposal \rangle$

```
                    If any proposal has been Failed, mark the transaction Failed.
                  ∨ ∧ ∃ t ∈ transaction[i].targets :
                        ∧ proposal[t][i].phase  = Initialize
                        ∧ proposal[t][i].status = Failed
                    ∧ transaction' = [transaction EXCEPT ![i].status = Failed]
                    ∧ UNCHANGED ⟨proposal⟩
              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.
            ∨ ∧ transaction[i].status = Complete
              ∧ ∀ t ∈ transaction[i].targets :
                  ∧ proposal[t][i].dependencyIndex > 0 ⇒
                        ∨ transaction[proposal[t][i].dependencyIndex].isolation ≠ Serializable
                        ∨ Phase[transaction[proposal[t][i].dependencyIndex].phase] > Phase[Validate]
                        ∨ ∧ transaction[proposal[t][i].dependencyIndex].phase = Validate
                          ∧ transaction[proposal[t][i].dependencyIndex].status ∈ {Complete, Failed}
              ∧ transaction' = [transaction EXCEPT ![i].phase  = Validate,
                                                   ![i].status = Pending]
              ∧ UNCHANGED ⟨proposal⟩
            ∨ ∧ transaction[i].status = Failed
              ∧ transaction' = [transaction EXCEPT ![i].phase  = Abort,
                                                   ![i].status = Pending]
              ∧ UNCHANGED ⟨proposal⟩
        ∨ ∧ transaction[i].phase = Validate
          ∧ ∨ ∧ transaction[i].status = Pending
                Move the transaction's proposals to the Validating state
              ∧ ∨ ∧ ∃ t ∈ transaction[i].targets :
                        ∧ proposal[t][i].phase ≠ Validate
                        ∧ proposal' = [proposal EXCEPT ![t] = [
                                            proposal[t] EXCEPT ![i].phase  = Validate,
                                                               ![i].status = Pending]]
                  ∧ UNCHANGED ⟨transaction⟩
                If all proposals have been Complete, mark the transaction Complete.
                ∨ ∧ ∀ t ∈ transaction[i].targets :
                        ∧ proposal[t][i].phase  = Validate
                        ∧ proposal[t][i].status = Complete
                  ∧ transaction' = [transaction EXCEPT ![i].status = Complete]
                  ∧ UNCHANGED ⟨proposal⟩
                If any proposal has been Failed, mark the transaction Failed.
                ∨ ∧ ∃ t ∈ transaction[i].targets :
                        ∧ proposal[t][i].phase  = Validate
                        ∧ proposal[t][i].status = Failed
                  ∧ transaction' = [transaction EXCEPT ![i].status = 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].status = Complete
        ∧ ∀ t ∈ transaction[i].targets :
            ∧ proposal[t][i].dependencyIndex > 0 ⇒
                ∨ transaction[proposal[t][i].dependencyIndex].isolation ≠ Serializable
                ∨ Phase[transaction[proposal[t][i].dependencyIndex].phase] > Phase[Commit]
                ∨ ∧ transaction[proposal[t][i].dependencyIndex].phase = Commit
                  ∧ transaction[proposal[t][i].dependencyIndex].status ∈ {Complete, Failed}
        ∧ transaction' = [transaction EXCEPT ![i].phase  = Commit,
                                             ![i].status = Pending]
        ∧ UNCHANGED ⟨proposal⟩
      ∨ ∧ transaction[i].status = Failed
        ∧ transaction' = [transaction EXCEPT ![i].phase  = Abort,
                                             ![i].status = Pending]
        ∧ UNCHANGED ⟨proposal⟩
  ∨ ∧ transaction[i].phase = Commit
    ∧ ∨ ∧ transaction[i].status = Pending
          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].status = Pending]]
          ∧ UNCHANGED ⟨transaction⟩
          If all proposals have been Complete, mark the transaction Complete.
        ∨ ∧ ∀ t ∈ transaction[i].targets :
              ∧ proposal[t][i].phase  = Commit
              ∧ proposal[t][i].status = Complete
          ∧ transaction' = [transaction EXCEPT ![i].status = Complete]
          ∧ 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.
    ∨ ∧ transaction[i].status = Complete
      ∧ ∀ t ∈ transaction[i].targets :
          ∧ proposal[t][i].dependencyIndex > 0 ⇒
              ∨ transaction[proposal[t][i].dependencyIndex].isolation ≠ Serializable
              ∨ Phase[transaction[proposal[t][i].dependencyIndex].phase] > Phase[Apply]
              ∨ ∧ transaction[proposal[t][i].dependencyIndex].phase = Apply
                ∧ transaction[proposal[t][i].dependencyIndex].status ∈ {Complete, Failed}
      ∧ transaction' = [transaction EXCEPT ![i].phase  = Apply,
```

```
                                                    ![i].status  = Pending]
            ∧ UNCHANGED ⟨proposal⟩
      ∨ ∧ transaction[i].phase = Apply
        ∧ transaction[i].status = Pending
            Move the transaction's proposals to the Applying state
        ∧ ∨ ∧ ∃ t ∈ transaction[i].targets :
                  ∧ proposal[t][i].phase ≠ Validate
                  ∧ proposal' = [proposal EXCEPT ![t] = [
                                    proposal[t] EXCEPT ![i].phase  = Apply,
                                                       ![i].status = Pending]]
            ∧ UNCHANGED ⟨transaction⟩
            If all proposals have been Complete, mark the transaction Complete.
          ∨ ∧ ∀ t ∈ transaction[i].targets :
                  ∧ proposal[t][i].phase  = Apply
                  ∧ proposal[t][i].status = Complete
            ∧ transaction' = [transaction EXCEPT ![i].status = Complete]
            ∧ UNCHANGED ⟨proposal⟩
            If any proposal has been Failed, mark the transaction Failed.
          ∨ ∧ ∃ t ∈ transaction[i].targets :
                  ∧ proposal[t][i].phase  = Apply
                  ∧ proposal[t][i].status = Failed
            ∧ transaction' = [transaction EXCEPT ![i].status = Failed]
            ∧ UNCHANGED ⟨proposal⟩
      The Aborting state is used to clean up transactions that have failed during
      the Initializing or Validating phases.
      ∨ ∧ transaction[i].phase = Abort
        ∧ transaction[i].status = Pending
            Move the transaction's proposals to the Aborting state
        ∧ ∨ ∧ ∃ t ∈ transaction[i].targets :
                  ∧ proposal[t][i].phase ≠ Validate
                  ∧ proposal' = [proposal EXCEPT ![t] = [
                                    proposal[t] EXCEPT ![i].phase  = Abort,
                                                       ![i].status = Pending]]
            ∧ UNCHANGED ⟨transaction⟩
            If all proposals have been Complete, mark the transaction Complete.
          ∨ ∧ ∀ t ∈ transaction[i].targets :
                  ∧ proposal[t][i].phase  = Abort
                  ∧ proposal[t][i].status = Complete
            ∧ transaction' = [transaction EXCEPT ![i].status = Complete]
            ∧ UNCHANGED ⟨proposal⟩
  ∧ UNCHANGED ⟨configuration, mastership, target⟩

Reconcile a proposal
ReconcileProposal(n, t, i) ≜
  ∧ ∨ ∧ proposal[t][i].phase = Initialize
```

$\land\ proposal[t][i].status = Pending$
$\land\ proposal' = [proposal \text{ EXCEPT } ![t] = [$
$\quad proposal[t] \text{ EXCEPT } ![i] = [$
$\quad\quad status \mapsto Complete,$
$\quad\quad dependencyIndex \mapsto configuration[t].proposedIndex] \mathbin{@@} proposal[t][i]]]$
$\land\ configuration' = [configuration \text{ EXCEPT } ![t].proposedIndex = i]$
$\land\ \text{UNCHANGED } \langle target \rangle$

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.

$\lor\ \land\ proposal[t][i].phase = Validate$
$\land\ proposal[t][i].status = Pending$
$\land\ configuration[t].committedIndex = proposal[t][i].dependencyIndex$

For *Change* proposals validate the set of requested changes.

$\land\ \lor\ \land\ proposal[t][i].type = Change$
$\land\ \text{LET } rollbackIndex \triangleq configuration[t].configIndex$
$rollbackValues \triangleq [p \in \text{DOMAIN } proposal[t][i].values \mapsto$
$\text{IF } p \in \text{DOMAIN } configuration[t].values \text{ THEN}$
$configuration[t].values[p]$
ELSE
$[value \mapsto Nil,$
$delete \mapsto \text{TRUE}]]$

Model validation successes and failures with *Valid* and *Invalid* results.

$\text{IN}\quad \exists\, r \in \{Valid, Invalid\} :$

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.

$\lor\ \land\ r = Valid$
$\land\ proposal' = [proposal \text{ EXCEPT } ![t] = [$
$proposal[t] \text{ EXCEPT } ![i].rollbackIndex = rollbackIndex,$
$![i].rollbackValues = rollbackValues,$
$![i].status = Complete]]$
$\lor\ \land\ r = Invalid$
$\land\ proposal' = [proposal \text{ EXCEPT } ![t] = [$
$proposal[t] \text{ EXCEPT } ![i].status = Failed]]$

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.

$\land\ \lor\ \land\ configuration[t].configIndex = proposal[t][i].rollback$
$\land\ \text{LET } rollbackIndex \triangleq proposal[t][proposal[t][i].rollback].rollbackIndex$
$rollbackValues \triangleq proposal[t][proposal[t][i].rollback].rollbackValues$

$$
\begin{array}{l}
\text{IN}\quad \exists\, r \in \{\mathit{Valid},\ \mathit{Invalid}\} : \\
\quad \text{If the } \mathit{Rollback} \text{ is } \mathit{Valid}\text{, record the changes required to roll back the target proposal and the index to which the configuration is being rolled back.} \\
\quad \lor\ \land\ r = \mathit{Valid} \\
\qquad \land\ \mathit{proposal}' = [\mathit{proposal}\ \text{EXCEPT}\ ![t] = [\\
\qquad\qquad \mathit{proposal}[t]\ \text{EXCEPT}\ ![i].\mathit{rollbackIndex} = \mathit{rollbackIndex}, \\
\qquad\qquad\qquad\qquad\qquad\quad ![i].\mathit{rollbackValues} = \mathit{rollbackValues}, \\
\qquad\qquad\qquad\qquad\qquad\quad ![i].\mathit{status} = \mathit{Complete}]] \\
\quad \lor\ \land\ r = \mathit{Invalid} \\
\qquad \land\ \mathit{proposal}' = [\mathit{proposal}\ \text{EXCEPT}\ ![t] = [\\
\qquad\qquad\qquad \mathit{proposal}[t]\ \text{EXCEPT}\ ![i].\mathit{status} = \mathit{Failed}]] \\
\quad \text{If the } \mathit{Rollback} \text{ target is not the most recent change to the configuration, fail validation for the proposal.} \\
\quad \lor\ \land\ \mathit{configuration}[t].\mathit{configIndex} \neq \mathit{proposal}[t][i].\mathit{rollback} \\
\qquad \land\ \mathit{proposal}' = [\mathit{proposal}\ \text{EXCEPT}\ ![t] = [\mathit{proposal}[t]\ \text{EXCEPT}\ ![i].\mathit{status} = \mathit{Failed}]] \\
\quad \text{If a } \mathit{Rollback} \text{ proposal is attempting to roll back another } \mathit{Rollback}\text{, fail validation for the proposal.} \\
\quad \lor\ \land\ \mathit{proposal}[t][\mathit{proposal}[t][i].\mathit{rollback}].\mathit{type} = \mathit{Rollback} \\
\qquad \land\ \mathit{proposal}' = [\mathit{proposal}\ \text{EXCEPT}\ ![t] = [\\
\qquad\qquad \mathit{proposal}[t]\ \text{EXCEPT}\ ![i].\mathit{status} = \mathit{Failed}]] \\
\land\ \text{UNCHANGED}\ \langle \mathit{configuration},\ \mathit{target} \rangle \\
\text{While in the } \mathit{Commit} \text{ state, commit the proposed changes to the configuration.} \\
\lor\ \land\ \mathit{proposal}[t][i].\mathit{phase} = \mathit{Commit} \\
\land\ \mathit{proposal}[t][i].\mathit{status} = \mathit{Pending} \\
\quad \text{Only commit the proposal if the prior proposal has already been committed.} \\
\land\ \mathit{configuration}[t].\mathit{committedIndex} = \mathit{proposal}[t][i].\mathit{dependencyIndex} \\
\quad \text{If the proposal is a change, commit the change values and set the configuration index to the proposal index.} \\
\land\ \lor\ \land\ \mathit{proposal}[t][i].\mathit{type} = \mathit{Change} \\
\qquad \land\ \mathit{configuration}' = [\mathit{configuration}\ \text{EXCEPT}\ ![t].\mathit{values} = \mathit{proposal}[t][i].\mathit{values}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad ![t].\mathit{configIndex} = i, \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad ![t].\mathit{committedIndex} = i] \\
\quad \text{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.} \\
\quad\ \lor\ \land\ \mathit{proposal}[t][i].\mathit{type} = \mathit{Rollback} \\
\qquad \land\ \mathit{configuration}' = [\mathit{configuration}\ \text{EXCEPT}\ ![t].\mathit{values} = \mathit{proposal}[t][i].\mathit{rollbackValues}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad ![t].\mathit{configIndex} = \mathit{proposal}[t][i].\mathit{rollbackIndex}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad ![t].\mathit{committedIndex} = i] \\
\land\ \mathit{proposal}' = [\mathit{proposal}\ \text{EXCEPT}\ ![t] = [\mathit{proposal}[t]\ \text{EXCEPT}\ ![i].\mathit{status} = \mathit{Complete}]] \\
\land\ \text{UNCHANGED}\ \langle \mathit{target} \rangle \\
\text{While in the } \mathit{Apply} \text{ phase, apply the proposed changes to the target.} \\
\lor\ \land\ \mathit{proposal}[t][i].\mathit{phase} = \mathit{Apply} \\
\land\ \mathit{proposal}[t][i].\mathit{status} = \mathit{Pending}
\end{array}
$$

$\land$ $configuration[t].appliedIndex = proposal[t][i].dependencyIndex$
$\land$ $configuration[t].appliedTerm = mastership[t].term$
$\land$ $mastership[t].master = n$
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.
$\land \lor \land proposal[t][i].type = Change$
$\land\ target' = [target$ EXCEPT $![t] = proposal[t][i].values$ @@ $target[t]]$
$\land\ configuration' = [configuration$ 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.
$\lor \land proposal[t][i].type = Rollback$
$\land\ target' = [target$ EXCEPT $![t] = proposal[t][i].rollbackValues$ @@ $target[t]]$
$\land\ configuration' = [configuration$ EXCEPT
$![t].appliedIndex\ \ = i,$
$![t].appliedValues = proposal[t][i].rollbackValues$ @@ $configuration[t].appliedValues]$
$\land\ proposal' = [proposal$ EXCEPT $![t] = [proposal[t]$ EXCEPT $![i].status = Complete]]$
If the proposal could not be applied, update the configuration's applied index
and mark the proposal *Failed*.
$\lor \land r = Failure$
$\land\ configuration' = [configuration$ EXCEPT $![t].appliedIndex = i]$
$\land\ proposal' = [proposal$ EXCEPT $![t] = [proposal[t]$ EXCEPT $![i].status = Failed]]$
$\land$ UNCHANGED $\langle target \rangle$
$\lor \land proposal[t][i].phase = Abort$
$\land\ proposal[t][i].status = Pending$
The *committedIndex* will always be greater than or equal to the *appliedIndex*.
If only the *committedIndex* matches the proposal's *dependencyIndex*, update
the *committedIndex* to enable commits of later proposals, but do not
mark the *Abort* phase *Complete* until the *appliedIndex* has been incremented.
$\land \lor \land configuration[t].committedIndex = proposal[t][i].dependencyIndex$
$\land\ configuration[t].appliedIndex \neq proposal[t][i].dependencyIndex$
$\land\ configuration' = [configuration$ EXCEPT $![t].committedIndex = i]$
$\land$ UNCHANGED $\langle proposal \rangle$
If both the configuration's *committedIndex* and *appliedIndex* match the
proposal's *dependencyIndex*, update the *committedIndex* and *appliedIndex*
and mark the proposal *Complete* for the *Abort* phase.
$\lor \land configuration[t].committedIndex = proposal[t][i].dependencyIndex$
$\land\ configuration[t].appliedIndex = proposal[t][i].dependencyIndex$
$\land\ configuration' = [configuration$ EXCEPT $![t].committedIndex = i,$
$![t].appliedIndex\ \ = i]$
$\land\ proposal' = [proposal$ EXCEPT $![t] = [proposal[t]$ EXCEPT $![i].status = Complete]]$

$\land$ UNCHANGED $\langle target \rangle$
$\land$ UNCHANGED $\langle transaction, mastership \rangle$

This section models the Configuration reconciler.

$ReconcileConfiguration(n, t) \triangleq$
$\land \lor \land Target[t].persistent$
$\land configuration[t].status \neq Complete$
$\land configuration' = [configuration$ EXCEPT $![t].status = Complete]$
$\land$ UNCHANGED $\langle target \rangle$
$\lor \land \neg Target[t].persistent$
$\land \lor mastership[t].term > configuration[t].configTerm$
$\lor \land mastership[t].term = configuration[t].configTerm$
$\land mastership[t].master = Nil$
$\land configuration' = [configuration$ EXCEPT $![t].configTerm = mastership[t].term,$
$![t].status = Pending]$
$\land$ UNCHANGED $\langle target \rangle$
$\lor \land configuration[t].status = Pending$
$\land mastership[t].term = configuration[t].configTerm$
$\land mastership[t].master = n$
$\land target' = [target$ EXCEPT $![t] = configuration[t].appliedValues]$
$\land configuration' = [configuration$ EXCEPT $![t].appliedTerm = mastership[t].term,$
$![t].status = Complete]$
$\land$ UNCHANGED $\langle proposal, transaction, mastership \rangle$

Formal specification, constraints, and theorems.

$Init \triangleq$
$\land transaction = \langle \rangle$
$\land proposal = [t \in$ DOMAIN $Target \mapsto$
$[p \in \{\} \mapsto [phase \mapsto Initialize,$
$status \mapsto Pending]]]$
$\land configuration = [t \in$ DOMAIN $Target \mapsto$
$[target \mapsto t,$
$status \mapsto Pending,$
$values \mapsto$
$[path \in \{\} \mapsto$
$[path \mapsto path,$
$value \mapsto Nil,$
$index \mapsto 0,$
$deleted \mapsto$ FALSE$]],$
$configIndex \mapsto 0,$
$configTerm \mapsto 0,$
$proposedIndex \mapsto 0,$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{committedIndex} \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{appliedIndex} \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{appliedTerm} \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{appliedValues} \mapsto \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad [\mathit{path} \in \{\} \mapsto \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\mathit{path} \mapsto \mathit{path}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\; \mathit{value} \mapsto \mathit{Nil}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\; \mathit{index} \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\; \mathit{deleted} \mapsto \text{FALSE}]]]] \\
\quad \land\ \mathit{target} = [t \in \text{DOMAIN } \mathit{Target} \mapsto \\
\qquad\qquad\qquad [\mathit{path} \in \{\} \mapsto \\
\qquad\qquad\qquad\quad [\mathit{value} \mapsto \mathit{Nil}]]] \\
\quad \land\ \mathit{mastership} = [t \in \text{DOMAIN } \mathit{Target} \mapsto [\mathit{master} \mapsto \mathit{Nil},\ \mathit{term} \mapsto 0]]
\end{array}
$$

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

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

$$
\begin{array}{l}
\mathit{Order} \triangleq \\
\quad \land\ \forall\, i,\ j \in \text{DOMAIN } \mathit{transaction} : \\
\qquad\quad \lor\ j \leq i \\
\qquad\quad \lor\ \mathit{Phase}[\mathit{transaction}[i].\mathit{phase}] \geq \mathit{Phase}[\mathit{transaction}[j].\mathit{phase}] \\
\qquad\quad \lor\ \mathit{transaction}[i].\mathit{targets} \cap \mathit{transaction}[j].\mathit{targets} = \{\} \\
\qquad\quad \lor\ \mathit{transaction}[j].\mathit{status} \in \{\mathit{Pending},\ \mathit{Failed}\} \\
\quad \land\ \forall\, t \in \text{DOMAIN } \mathit{proposal} : \\
\qquad\quad \forall\, i,\ j \in \text{DOMAIN } \mathit{proposal}[t] : \\
\qquad\qquad \lor\ j \leq i
\end{array}
$$

```
        ∨ Phase[proposal[t][i].phase] ≥ Phase[proposal[t][j].phase]
        ∨ proposal[t][j].status ∈ {Pending, Failed}

Consistency ≜
  ∀ t ∈ DOMAIN target :
    LET
        Compute the transaction indexes that have been applied to the target
        appliedIndexes ≜ {i ∈ DOMAIN transaction :
                              ∧ transaction[i].type = Change
                              ∧ i ∈ DOMAIN proposal[t]
                              ∧ proposal[t][i].phase = Apply
                              ∧ proposal[t][i].status = 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].status = Complete}
        Compute the set of paths in the target that have been updated by transactions
        appliedPaths ≜ UNION {DOMAIN transaction[i].values[t] : i ∈ appliedIndexes}
        Compute the highest index applied to the target for each path
        pathIndexes ≜ [p ∈ appliedPaths ↦ CHOOSE i ∈ appliedIndexes :
                                              ∀ j ∈ appliedIndexes :
                                                  ∧ 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, j ∈ DOMAIN transaction :
      ∨ j ≤ i
      ∨ transaction[i].targets ∩ transaction[j].targets = {}
      ∨ transaction[i].isolation ≠ Serializable
      ∨ ∧ ∨ ∧ transaction[i].phase ∈ {Commit, Abort}
              ∧ transaction[i].status ∈ {Complete, Failed}
            ∨ Phase[transaction[i].phase] > Phase[Commit]
            ∨ Phase[transaction[j].phase] < Phase[Commit]
        ∧ ∨ ∧ transaction[i].phase ∈ {Apply, Abort}
              ∧ transaction[i].status ∈ {Complete, Failed}
            ∨ Phase[transaction[j].phase] < Phase[Apply]
      ∨ transaction[j].status = Failed
```

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

$Completion \triangleq$
$\land \forall i \in$ DOMAIN $transaction$:
$\land transaction[i].phase = Commit$
$\land transaction[i].status \in \{Complete, Failed\}$
$\land \forall i \in$ DOMAIN $transaction$:
$\land transaction[i].phase = Apply$
$\land transaction[i].status \in \{Complete, Failed\}$
$\land \forall t \in$ DOMAIN $proposal$:
$\forall i \in$ DOMAIN $proposal[t]$:
$\land proposal[t][i].phase = Commit$
$\land proposal[t][i].status \in \{Complete, Failed\}$
$\land \forall t \in$ DOMAIN $proposal$:
$\forall i \in$ DOMAIN $proposal[t]$:
$\land proposal[t][i].phase = Apply$
$\land proposal[t][i].status \in \{Complete, Failed\}$

THEOREM $Liveness \triangleq Spec \Rightarrow \Diamond Completion$

Type assumptions.

ASSUME $Nil \in$ STRING

ASSUME $\forall phase \in$ DOMAIN $Phase : phase \in$ STRING

ASSUME $\forall status \in$ DOMAIN $Status : status \in$ STRING

ASSUME $\land IsFiniteSet(Node)$
$\land \forall n \in Node$:
$\land n \notin$ DOMAIN $Target$
$\land n \in$ STRING

ASSUME $\land \forall t \in$ DOMAIN $Target$:
$\land t \notin Node$
$\land t \in$ STRING
$\land Target[t].persistent \in$ BOOLEAN
$\land \forall p \in$ DOMAIN $Target[t].values$:
$IsFiniteSet(Target[t].values[p])$

---

* Modification History
* Last modified *Mon Feb* 07 14:58:32 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*