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

$Status \triangleq$
$\langle Initializing,$
$Initialized,$
$Validating,$
$Validated,$
$Committing,$
$Committed,$

Applying,
Applied,
Failed⟩

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*

$Phase(s) \triangleq$ CHOOSE $i \in$ DOMAIN $Status : Status[i] = s$

ASSUME $Nil \in$ STRING

ASSUME $Pending \in$ STRING
ASSUME $Initializing \in$ STRING
ASSUME $Initialized \in$ STRING
ASSUME $Validating \in$ STRING
ASSUME $Validated \in$ STRING
ASSUME $Committing \in$ STRING
ASSUME $Committed \in$ STRING
ASSUME $Applying \in$ STRING
ASSUME $Applied \in$ STRING
ASSUME $Synchronizing \in$ STRING
ASSUME $Synchronized \in$ STRING
ASSUME $Persisted \in$ STRING
ASSUME $Failed \in$ STRING

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

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

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

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

$$
\begin{array}{l}
Value(s,\ t,\ p) \triangleq \\
\quad \text{LET}\ value \triangleq \text{CHOOSE}\ v \in s : v.target = t \land v.path = p \\
\quad \text{IN} \\
\qquad [value\ \mapsto value.value, \\
\qquad\ delete \mapsto value.delete]
\end{array}
$$

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

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

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

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

$$
\begin{array}{l}
ValidTargets \triangleq \\
\quad \text{UNION}\ \{\{p\ @@\ [target \mapsto t] : p \in\ ValidPaths(t)\} : t \in \text{DOMAIN}\ \ Target\}
\end{array}
$$

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.

$$
\begin{array}{l}
ValidChanges \triangleq \\
\quad \text{LET}\ changeSets \triangleq \{s \in \text{SUBSET}\ \ ValidTargets : \\
\qquad\qquad\qquad\qquad\qquad \forall\, t \in \text{DOMAIN}\ \ Target\quad :
\end{array}
$$

$\forall\, p \in \text{DOMAIN } Target[t].values :$
$Cardinality(\{v \in s : v.target = t \land v.path = p\}) \leq 1\}$

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 $transaction :$
$\forall\, j \in \text{DOMAIN } transaction : i \geq j$
IN $i + 1$

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

$RequestChange(c) \triangleq$

$\land \exists\, isolation \in \{ReadCommitted,\ Serializable\} :$
$\land\ transaction' = transaction \mathbin{@@} (NextIndex :> [type \mapsto Change,$
$index \mapsto NextIndex,$
$isolation \mapsto isolation,$
$values \mapsto c,$
$targets \mapsto \{\},$
$status \mapsto Initializing])$
$\land$ UNCHANGED $\langle proposal,\ configuration,\ mastership,\ target\rangle$

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

$RequestRollback(t) \triangleq$

$\land \exists\, isolation \in \{ReadCommitted,\ Serializable\} :$
$\land\ transaction' = transaction \mathbin{@@} (NextIndex :> [type \mapsto Rollback,$
$index \mapsto NextIndex,$
$isolation \mapsto isolation,$
$rollback \mapsto t,$
$targets \mapsto \{\},$
$status \mapsto Initializing])$
$\land$ UNCHANGED $\langle proposal,\ configuration,\ mastership,\ target\rangle$

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 ↦ 0,
                                                    rollbackIndex   ↦ 0,
                                                    rollbackValues  ↦ ⟨⟩,
                                                    status          ↦ Initializing])
                                ELSE
                                  proposal[t]]
                          ∨ ∧ ∨ ∧ transaction[i].rollback ∈ DOMAIN transaction
                                ∧ transaction[transaction[i].rollback].type = Rollback
                              ∨ transaction[i].rollback ∉ DOMAIN transaction
                            ∧ transaction' = [transaction EXCEPT ![i].status = Failed]
                            ∧ UNCHANGED ⟨proposal⟩
                    ∨ ∧ transaction[i].targets ≠ {}
                        If all proposals have been Initialized, mark the transaction Initialized.
                      ∧ ∨ ∧ ∀ t ∈ transaction[i].targets : proposal[t][i].status = Initialized
                          ∧ transaction' = [transaction EXCEPT ![i].status = Initialized]
                          ∧ 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⟩
    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 = Initialized
  ∧ ∀ t ∈ transaction[i].targets :
      proposal[t][i].dependencyIndex ≠ 0 ⇒
        (transaction[proposal[t][i].dependencyIndex].isolation = Serializable ⇒
          Phase(transaction[proposal[t][i].dependencyIndex].status) ≥ Phase(Validated))
  ∧ transaction' = [transaction EXCEPT ![i].status = Validating]
  ∧ UNCHANGED ⟨proposal⟩
∨ ∧ transaction[i].status = Validating
    Move the transaction's proposals to the Validating state
  ∧ ∨ ∧ ∃ t ∈ transaction[i].targets : Phase(proposal[t][i].status) < Phase(Validating)
      ∧ proposal' = [t ∈ DOMAIN proposal ↦
                       IF t ∈ transaction[i].targets THEN
                         [proposal[t] EXCEPT ![i].status = Validating]
                       ELSE
                         proposal[t]]
      ∧ UNCHANGED ⟨transaction⟩
    If all proposals have been Validated, mark the transaction Validated.
    ∨ ∧ ∀ t ∈ transaction[i].targets : proposal[t][i].status = Validated
      ∧ transaction' = [transaction EXCEPT ![i].status = Validated]
      ∧ UNCHANGED ⟨proposal⟩
    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

$\land \lor \land \exists\, t \in transaction[i].targets : Phase(proposal[t][i].status) < Phase(Committing)$
$\land proposal' = [t \in \text{DOMAIN } proposal \mapsto$
$\text{IF } t \in transaction[i].targets \text{ THEN}$
$[proposal[t] \text{ EXCEPT } ![i].status = Committing]$
ELSE
$proposal[t]]$
$\land \text{UNCHANGED } \langle transaction \rangle$

If all proposals have been *Committed*, mark the transaction *Committed*.

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

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 *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

$\land \lor \land \exists\, t \in transaction[i].targets : Phase(proposal[t][i].status) < Phase(Applying)$

```
                    ∧ proposal′ = [t ∈ DOMAIN proposal ↦
                                    IF t ∈ transaction[i].targets THEN
                                      [proposal[t] EXCEPT ![i].status = Applying]
                                    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⟩
              ∨ ∧ transaction[i].status = Applied
           ∧ 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 ↦ [
                                     p ↦ IF p ∈ DOMAIN configuration[t].values THEN
                                           configuration[t].values[p]
                                         ELSE
                                           [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] = [
```

$proposal[t] \text{ EXCEPT } ![i].rollbackIndex = rollbackIndex,$
$![i].rollbackValues = rollbackValues,$
$![i].status = Validated]]$
$\land \text{ UNCHANGED } \langle configuration \rangle$
$\lor \land r = Invalid$
$\land configuration' = [configuration \text{ EXCEPT } ![t].committedIndex = i]$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [$
$proposal[t] \text{ EXCEPT } ![i].status = Failed]]$

For *Rollback* proposals, validate the rollback changes which are
proposal being rolled back.

$\lor \land proposal[t][i].type = Rollback$
$\land \lor \land configuration[t].configIndex = proposal[t][i].rollback$
$\land \lor \land proposal[t][proposal[t][i].rollback].type = Change$
$\land \text{ LET } rollbackIndex \triangleq proposal[t][proposal[t][i].rollback].rollbackIndex$
$rollbackValues \triangleq proposal[t][proposal[t][i].rollback].rollbackValues$
$\text{IN } \exists r \in \{Valid, Invalid\} :$

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

$\lor \land proposal[t][proposal[t][i].rollabck].type = Rollback$
$\land configuration' = [configuration \text{ EXCEPT } ![t].committedIndex = i]$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [$
$proposal[t] \text{ EXCEPT } ![i].status = Failed]]$

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

$\lor \land configuration[t].configIndex \neq proposal[t][i].rollback$
$\land configuration' = [configuration \text{ EXCEPT } ![t].committedIndex = i]$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].status = Failed]]$
$\land \text{ UNCHANGED } \langle target \rangle$

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
                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
                  Model successful and failed target update requests.
                ∧ ∃ r ∈ {Success, Failure} :
                    ∨ ∧ r = Success
                      ∧ target' = [target EXCEPT ![t] = proposal[t][i].values @@ target[t]]
                      ∧ configuration' = [configuration EXCEPT
                            ![t].appliedIndex = i,
                            ![t].appliedValues = proposal[t][i].values @@ configuration[i].appliedValues]
                      ∧ proposal' = [proposal EXCEPT ![t] = [proposal[t] EXCEPT ![i].status = Applied]]
                    ∨ ∧ r = Failure
                      ∧ configuration' = [configuration EXCEPT ![t].appliedIndex = i]
                      ∧ proposal' = [proposal EXCEPT ![t] = [proposal[t] EXCEPT ![i].status = Failed]]
                      ∧ UNCHANGED ⟨target⟩
                If the proposal is a rollback, apply the rollback values and update the
                configuration's applied index and values with the rollback index and values.
              ∨ ∧ proposal[t][i].type = Rollback
                  Model successful and failed target update requests.
                ∧ ∃ r ∈ {Success, Failure} :
                    ∨ ∧ r = Success
                      ∧ target' = [target EXCEPT ![t] = proposal[t][i].values @@ target[t]]
                      ∧ configuration' = [configuration EXCEPT
                            ![t].appliedIndex = proposal[t][i].rollbackIndex,
                            ![t].appliedValues = proposal[t][i].rollbackValues @@ configuration[i].appliedValues]
```

$\land\ proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].status = Applied]]$
$\lor\ \land\ r = Failure$
$\land\ configuration' = [configuration \text{ EXCEPT } ![t].appliedIndex = proposal[t][i].rollbackIndex]$
$\land\ proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].status = Failed]]$
$\land\ \text{UNCHANGED } \langle target \rangle$
$\land\ \text{UNCHANGED } \langle transaction, mastership \rangle$

This section models the Configuration reconciler.

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

Init and next state predicates

$Init \triangleq$
$\land\ transaction = \langle \rangle$
$\land\ proposal = [t \in \text{DOMAIN } Target \mapsto$
$[p \in \{\} \mapsto [status \mapsto Initializing]]]$
$\land\ configuration = [t \in \text{DOMAIN } Target \mapsto$
$[target \mapsto t,$
$status \mapsto Pending,$
$values \mapsto$
$[path \in \{\} \mapsto$

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

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

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

$Order \triangleq$ TRUE *TODO* redefine order spec

THEOREM $Safety \triangleq Spec \Rightarrow \Box Order$

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

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

* Modification History
* Last modified Sun *Feb* 06 13:24:46 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*