

---

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*  $\triangleq$

LET *phases*  $\triangleq$   $\langle$ *Initialize,*

*Validate,*

*Abort,*

*Commit,*

*Apply* $\rangle$

IN  $[p \in \{phases[i] : i \in \text{DOMAIN } phases\} \mapsto$

CHOOSE  $i \in \text{DOMAIN } phases : phases[i] = p]$

Status constants

CONSTANTS

*Pending,*

*Complete,*

*Failed*

*Status*  $\triangleq$

```

LET statuses  $\triangleq$   $\langle$ Pending,
               Complete,
               Failed $\rangle$ 
IN  [s  $\in$  {statuses[i] : i  $\in$  DOMAIN statuses}  $\mapsto$ 
      CHOOSE i  $\in$  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*  $\triangleq$  [  
     *target1*  $\mapsto$  [*persistent*  $\mapsto$  FALSE, *values*  $\mapsto$  [  
         *path1*  $\mapsto$  {"*value1*", "*value2*"},  
         *path2*  $\mapsto$  {"*value2*", "*value3*" }]],  
     *target2*  $\mapsto$  [*persistent*  $\mapsto$  TRUE, *values*  $\mapsto$  [  
         *path2*  $\mapsto$  {"*value3*", "*value4*" },  
         *path3*  $\mapsto$  {"*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*  $\in$   
     {*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  $\triangleq$  [
  type      ::= type  $\in$  Type,
  index     ::= index  $\in$  Nat,
  isolation ::= isolation  $\in$  {IsolationDefault, IsolationSerializable}
  values ::= [
    target  $\in$  SUBSET (DOMAIN Target)  $\mapsto$  [ path  $\in$  SUBSET (DOMAIN Target[target].values)  $\mapsto$ 
    [
      value ::= value  $\in$  STRING,
      delete ::= delete  $\in$  BOOLEAN ]],
    rollback ::= index  $\in$  Nat,
    targets ::= targets  $\in$  SUBSET (DOMAIN Target)
    phase     ::= phase  $\in$  Phase]
    status ::= status  $\in$  Status]

TYPE Proposal  $\triangleq$  [
  type      ::= type  $\in$  Type,
  index     ::= index  $\in$  Nat,
  values     ::= [ path  $\in$  SUBSET (DOMAIN Target[target].values)  $\mapsto$  [
    value ::= value  $\in$  STRING,
    delete ::= delete  $\in$  BOOLEAN ]],
    rollback     ::= index  $\in$  Nat,
    dependencyIndex ::= dependencyIndex  $\in$  Nat,
    rollbackIndex  ::= rollbackIndex  $\in$  Nat,
    rollbackValues ::= [ path  $\in$  SUBSET (DOMAIN Target[target].values)  $\mapsto$  [
      value ::= value  $\in$  STRING,
      delete ::= delete  $\in$  BOOLEAN ]],
    phase        ::= phase  $\in$  Phase]
    status        ::= status  $\in$  Status]

TYPE Configuration  $\triangleq$  [
  id      ::= id  $\in$  STRING,
  target  ::= target  $\in$  STRING,
  values  ::= [ path  $\in$  SUBSET (DOMAIN Target[target])  $\mapsto$  [
    value ::= value  $\in$  STRING,
    index ::= index  $\in$  Nat,
    deleted ::= delete  $\in$  BOOLEAN ]],
    configIndex ::= configIndex  $\in$  Nat,
    configTerm  ::= configTerm  $\in$  Nat,
    proposedIndex ::= proposedIndex  $\in$  Nat,
    committedIndex ::= committedIndex  $\in$  Nat,
    appliedIndex ::= appliedIndex  $\in$  Nat,
    appliedTerm  ::= appliedTerm  $\in$  Nat,

```

$$\begin{aligned} appliedValues ::= & [ path \in \text{SUBSET } (\text{DOMAIN } Target[target]) \mapsto [ \\ & value ::= value \in \text{STRING}, \\ & index ::= index \in \text{Nat}, \\ & deleted ::= delete \in \text{BOOLEAN} ]], \\ status ::= & status \in \text{Status} \end{aligned}$$

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 masterhips

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  $t$

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

$UnsetMaster(t) \triangleq$

$$\begin{aligned} &\wedge mastership[t].master \neq Nil \\ &\wedge mastership' = [mastership \text{ EXCEPT } ![t].master = Nil] \\ &\wedge \text{UNCHANGED } \langle transaction, proposal, configuration, target \rangle \end{aligned}$$

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

$$\begin{aligned} Value(s, t, p) &\triangleq \\ \text{LET } value &\triangleq \text{CHOOSE } v \in s : v.target = t \wedge v.path = p \\ \text{IN} \end{aligned}$$

$[value \mapsto value.value,$   
 $delete \mapsto value.delete]$

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

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

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

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

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

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.

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

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.

$NextIndex \triangleq$   
 $IF DOMAIN transaction = \{ \} THEN$   
 $1$   
 $ELSE$   
 $LET i \triangleq CHOOSE i \in DOMAIN transaction :$   
 $\quad \forall j \in DOMAIN transaction : i \geq j$   
 $IN i + 1$

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

$RequestChange(c) \triangleq$   
 $\wedge \exists isolation \in \{ ReadCommitted, Serializable \} :$   
 $\quad \wedge transaction' = transaction @@@ (NextIndex :> [type \mapsto Change,$   
 $\quad \quad \quad index \mapsto NextIndex,$   
 $\quad \quad \quad isolation \mapsto isolation,$   
 $\quad \quad \quad values \mapsto c,$   
 $\quad \quad \quad targets \mapsto \{ \},$

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


---

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

Transactions 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

$\text{ReconcileTransaction}(n, i) \triangleq$

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.

$\wedge \vee \wedge \text{transaction}[i].\text{phase} = \text{Initialize}$   
 $\wedge \vee \wedge \text{transaction}[i].\text{status} = \text{Pending}$

Serialize transaction initialization

$\wedge i - 1 \in \text{DOMAIN } \text{transaction} \Rightarrow$   
 $\vee \text{Phase}[\text{transaction}[i - 1].\text{phase}] > \text{Phase}[\text{Initialize}]$   
 $\vee \text{transaction}[i - 1].\text{status} \neq \text{Pending}$

If the transaction's targets are not yet set, create proposals and add targets to the transaction state.

$\wedge \vee \wedge \text{transaction}[i].\text{targets} = \{\}$

If the transaction is a change, the targets are taken from the change values.

$$\begin{aligned}
& \wedge \vee \wedge \text{transaction}[i].\text{type} = \text{Change} \\
& \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{targets} = \text{DOMAIN } \text{transaction}[i].\text{values}] \\
& \wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto \\
& \quad \text{IF } t \in \text{DOMAIN } \text{transaction}[i].\text{values} \text{ THEN} \\
& \quad \quad \text{proposal}[t] @ @ (i :> [type \mapsto \text{Change}, \\
& \quad \quad \quad \text{index} \mapsto i, \\
& \quad \quad \quad \text{values} \mapsto \text{transaction}[i].\text{values}[t], \\
& \quad \quad \quad \text{dependencyIndex} \mapsto 0, \\
& \quad \quad \quad \text{rollbackIndex} \mapsto 0, \\
& \quad \quad \quad \text{rollbackValues} \mapsto \langle \rangle, \\
& \quad \quad \quad \text{phase} \mapsto \text{Initialize}, \\
& \quad \quad \quad \text{status} \mapsto \text{Pending}]) \\
& \quad \text{ELSE} \\
& \quad \quad \text{proposal}[t] \\
& \quad \text{If the transaction is a rollback, the targets affected are} \\
& \quad \text{the targets of the change transaction being rolled back.} \\
& \vee \wedge \text{transaction}[i].\text{type} = \text{Rollback} \\
& \wedge \vee \wedge \text{transaction}[i].\text{rollback} \in \text{DOMAIN } \text{transaction} \\
& \wedge \text{transaction}[\text{transaction}[i].\text{rollback}].\text{type} = \text{Change} \\
& \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{targets} = \\
& \quad \text{DOMAIN } \text{transaction}[\text{transaction}[i].\text{rollback}].\text{values}] \\
& \wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto \\
& \quad \text{IF } t \in \text{DOMAIN } \text{transaction}[\text{transaction}[i].\text{rollback}].\text{values} \text{ THEN} \\
& \quad \quad \text{proposal}[t] @ @ (i :> [type \mapsto \text{Rollback}, \\
& \quad \quad \quad \text{index} \mapsto i, \\
& \quad \quad \quad \text{rollback} \mapsto \text{transaction}[i].\text{rollback}, \\
& \quad \quad \quad \text{dependencyIndex} \mapsto 0, \\
& \quad \quad \quad \text{rollbackIndex} \mapsto 0, \\
& \quad \quad \quad \text{rollbackValues} \mapsto \langle \rangle, \\
& \quad \quad \quad \text{phase} \mapsto \text{Initialize}, \\
& \quad \quad \quad \text{status} \mapsto \text{Pending}]) \\
& \quad \text{ELSE} \\
& \quad \quad \text{proposal}[t] \\
& \vee \wedge \vee \wedge \text{transaction}[i].\text{rollback} \in \text{DOMAIN } \text{transaction} \\
& \wedge \text{transaction}[\text{transaction}[i].\text{rollback}].\text{type} = \text{Rollback} \\
& \vee \text{transaction}[i].\text{rollback} \notin \text{DOMAIN } \text{transaction} \\
& \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{status} = \text{Failed}] \\
& \wedge \text{UNCHANGED } \langle \text{proposal} \rangle \\
& \vee \wedge \text{transaction}[i].\text{targets} \neq \{\} \\
& \quad \text{If all proposals have been Complete, mark the transaction Complete.} \\
& \wedge \vee \wedge \forall t \in \text{transaction}[i].\text{targets} : \\
& \quad \wedge \text{proposal}[t][i].\text{phase} = \text{Initialize} \\
& \quad \wedge \text{proposal}[t][i].\text{status} = \text{Complete} \\
& \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{status} = \text{Complete}] \\
& \wedge \text{UNCHANGED } \langle \text{proposal} \rangle
\end{aligned}$$

[illegible]



$\wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{status} = \text{Failed}]$   
 $\wedge \text{UNCHANGED } \langle \text{proposal} \rangle$

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.

$\vee \wedge \text{transaction}[i].\text{status} = \text{Complete}$   
 $\wedge \forall t \in \text{transaction}[i].\text{targets} :$   
 $\wedge \text{proposal}[t][i].\text{dependencyIndex} > 0 \Rightarrow$   
 $\vee \text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{isolation} \neq \text{Serializable}$   
 $\vee \text{Phase}[\text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{phase}] > \text{Phase}[\text{Commit}]$   
 $\vee \wedge \text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{phase} = \text{Commit}$   
 $\wedge \text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{status} \in \{\text{Complete}, \text{Failed}\}$   
 $\wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{phase} = \text{Commit},$   
 $\phantom{\wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } } ![i].\text{status} = \text{Pending}]$   
 $\wedge \text{UNCHANGED } \langle \text{proposal} \rangle$   
 $\vee \wedge \text{transaction}[i].\text{phase} = \text{Commit}$   
 $\wedge \vee \wedge \text{transaction}[i].\text{status} = \text{Pending}$

Move the transaction's proposals to the *Committing* state

$\wedge \vee \wedge \exists t \in \text{transaction}[i].\text{targets} : \text{proposal}[t][i].\text{phase} \neq \text{Commit}$   
 $\wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto$   
 $\phantom{\wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto}$  IF  $t \in \text{transaction}[i].\text{targets}$  THEN  
 $\phantom{\wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto}$   $[\text{proposal}[t] \text{ EXCEPT } ![i].\text{phase} = \text{Commit},$   
 $\phantom{\wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto}$   $![i].\text{status} = \text{Pending}]$   
 $\phantom{\wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto}$  ELSE  
 $\phantom{\wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto}$   $\text{proposal}[t]]$   
 $\wedge \text{UNCHANGED } \langle \text{transaction} \rangle$

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

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

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.

$\vee \wedge \text{transaction}[i].\text{status} = \text{Complete}$   
 $\wedge \forall t \in \text{transaction}[i].\text{targets} :$   
 $\wedge \text{proposal}[t][i].\text{dependencyIndex} > 0 \Rightarrow$   
 $\vee \text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{isolation} \neq \text{Serializable}$   
 $\vee \text{Phase}[\text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{phase}] > \text{Phase}[\text{Apply}]$   
 $\vee \wedge \text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{phase} = \text{Apply}$   
 $\wedge \text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{status} \in \{\text{Complete}, \text{Failed}\}$   
 $\wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{phase} = \text{Apply},$

$$\begin{aligned}
& \text{!}[i].status = Pending] \\
& \wedge \text{UNCHANGED } \langle proposal \rangle \\
& \vee \wedge transaction[i].status = Failed \\
& \wedge transaction' = [transaction \text{ EXCEPT } ![i].phase = Abort, \\
& \text{!}[i].status = Pending] \\
& \wedge \text{UNCHANGED } \langle proposal \rangle \\
& \vee \wedge transaction[i].phase = Apply \\
& \wedge transaction[i].status = Pending \\
& \text{Move the transaction's proposals to the Applying state} \\
& \wedge \vee \wedge \exists t \in transaction[i].targets : proposal[t][i].phase \neq Apply \\
& \wedge proposal' = [t \in \text{DOMAIN } proposal \mapsto \\
& \quad \text{IF } t \in transaction[i].targets \text{ THEN} \\
& \quad \quad [proposal[t] \text{ EXCEPT } ![i].phase = Apply, \\
& \quad \quad \text{!}[i].status = Pending] \\
& \quad \text{ELSE} \\
& \quad \quad proposal[t]] \\
& \wedge \text{UNCHANGED } \langle transaction \rangle \\
& \text{If all proposals have been Complete, mark the transaction Complete.} \\
& \vee \wedge \forall t \in transaction[i].targets : \\
& \quad \wedge proposal[t][i].phase = Apply \\
& \quad \wedge proposal[t][i].status = Complete \\
& \wedge transaction' = [transaction \text{ EXCEPT } ![i].status = Complete] \\
& \wedge \text{UNCHANGED } \langle proposal \rangle \\
& \text{If any proposal has been Failed, mark the transaction Failed.} \\
& \vee \wedge \exists t \in transaction[i].targets : \\
& \quad \wedge proposal[t][i].phase = Apply \\
& \quad \wedge proposal[t][i].status = Failed \\
& \wedge transaction' = [transaction \text{ EXCEPT } ![i].status = Failed] \\
& \wedge \text{UNCHANGED } \langle proposal \rangle \\
& \text{The Aborting state is used to clean up transactions that have failed during} \\
& \text{the Initializing or Validating phases.} \\
& \vee \wedge transaction[i].phase = Abort \\
& \wedge transaction[i].status = Pending \\
& \text{Move the transaction's proposals to the Aborting state} \\
& \wedge \vee \wedge \exists t \in transaction[i].targets : proposal[t][i].phase \neq Abort \\
& \wedge proposal' = [t \in \text{DOMAIN } proposal \mapsto \\
& \quad \text{IF } t \in transaction[i].targets \text{ THEN} \\
& \quad \quad [proposal[t] \text{ EXCEPT } ![i].phase = Abort, \\
& \quad \quad \text{!}[i].status = Pending] \\
& \quad \text{ELSE} \\
& \quad \quad proposal[t]] \\
& \wedge \text{UNCHANGED } \langle transaction \rangle \\
& \text{If all proposals have been Complete, mark the transaction Complete.} \\
& \vee \wedge \forall t \in transaction[i].targets : \\
& \quad \wedge proposal[t][i].phase = Abort
\end{aligned}$$

$\wedge \text{proposal}[t][i].\text{status} = \text{Complete}$   
 $\wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{status} = \text{Complete}]$   
 $\wedge \text{UNCHANGED } \langle \text{proposal} \rangle$   
 $\wedge \text{UNCHANGED } \langle \text{configuration}, \text{mastership}, \text{target} \rangle$

Reconcile a proposal

$\text{ReconcileProposal}(n, t, i) \triangleq$   
 $\wedge \vee \wedge \text{proposal}[t][i].\text{phase} = \text{Initialize}$   
 $\wedge \text{proposal}[t][i].\text{status} = \text{Pending}$   
 $\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [$   
 $\quad \text{proposal}[t] \text{ EXCEPT } ![i] = [$   
 $\quad \quad \text{status} \mapsto \text{Complete},$   
 $\quad \quad \text{dependencyIndex} \mapsto \text{configuration}[t].\text{proposedIndex}] @@ \text{proposal}[t][i]]$   
 $\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![t].\text{proposedIndex} = i]$   
 $\wedge \text{UNCHANGED } \langle \text{target} \rangle$

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.

$\vee \wedge \text{proposal}[t][i].\text{phase} = \text{Validate}$   
 $\wedge \text{proposal}[t][i].\text{status} = \text{Pending}$   
 $\wedge \text{configuration}[t].\text{committedIndex} = \text{proposal}[t][i].\text{dependencyIndex}$

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

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

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

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

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.

$\vee \wedge r = \text{Valid}$

$\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [$   
 $\quad \text{proposal}[t] \text{ EXCEPT } ![i].\text{rollbackIndex} = \text{rollbackIndex},$   
 $\quad \quad \quad ![i].\text{rollbackValues} = \text{rollbackValues},$   
 $\quad \quad \quad ![i].\text{status} = \text{Complete}]$

$\vee \wedge r = \text{Invalid}$

$\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [$   
 $\quad \text{proposal}[t] \text{ EXCEPT } ![i].\text{status} = \text{Failed}]$

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

$\vee \wedge \text{proposal}[t][i].\text{type} = \text{Rollback}$   
 Rollbacks can only be performed on *Change* type proposals.  
 $\wedge \vee \wedge \text{proposal}[t][\text{proposal}[t][i].\text{rollback}].\text{type} = \text{Change}$   
 Only roll back the change if it's the latest change made  
 to the configuration based on the configuration index.  
 $\wedge \vee \wedge \text{configuration}[t].\text{configIndex} = \text{proposal}[t][i].\text{rollback}$   
 $\wedge \text{LET } \text{rollbackIndex} \triangleq \text{proposal}[t][\text{proposal}[t][i].\text{rollback}].\text{rollbackIndex}$   
 $\text{rollbackValues} \triangleq \text{proposal}[t][\text{proposal}[t][i].\text{rollback}].\text{rollbackValues}$   
 IN  $\exists r \in \{ \text{Valid}, \text{Invalid} \} :$   
 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.  
 $\vee \wedge r = \text{Valid}$   
 $\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [$   
 $\text{proposal}[t] \text{ EXCEPT } ![i].\text{rollbackIndex} = \text{rollbackIndex},$   
 $![i].\text{rollbackValues} = \text{rollbackValues},$   
 $![i].\text{status} = \text{Complete}]$   
 $\vee \wedge r = \text{Invalid}$   
 $\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [$   
 $\text{proposal}[t] \text{ EXCEPT } ![i].\text{status} = \text{Failed}]$   
 If the *Rollback* target is not the most recent change to the configuration,  
 fail validation for the proposal.  
 $\vee \wedge \text{configuration}[t].\text{configIndex} \neq \text{proposal}[t][i].\text{rollback}$   
 $\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [\text{proposal}[t] \text{ EXCEPT } ![i].\text{status} = \text{Failed}]$   
 If a *Rollback* proposal is attempting to roll back another *Rollback*,  
 fail validation for the proposal.  
 $\vee \wedge \text{proposal}[t][\text{proposal}[t][i].\text{rollback}].\text{type} = \text{Rollback}$   
 $\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [$   
 $\text{proposal}[t] \text{ EXCEPT } ![i].\text{status} = \text{Failed}]$   
 $\wedge \text{UNCHANGED } \langle \text{configuration}, \text{target} \rangle$   
 While in the *Commit* state, commit the proposed changes to the configuration.  
 $\vee \wedge \text{proposal}[t][i].\text{phase} = \text{Commit}$   
 $\wedge \text{proposal}[t][i].\text{status} = \text{Pending}$   
 Only commit the proposal if the prior proposal has already been committed.  
 $\wedge \text{configuration}[t].\text{committedIndex} = \text{proposal}[t][i].\text{dependencyIndex}$   
 If the proposal is a change, commit the change values and set the configuration  
 index to the proposal index.  
 $\wedge \vee \wedge \text{proposal}[t][i].\text{type} = \text{Change}$   
 $\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![t].\text{values} = \text{proposal}[t][i].\text{values},$   
 $![t].\text{configIndex} = i,$   
 $![t].\text{committedIndex} = i]$   
 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.  
 $\vee \wedge \text{proposal}[t][i].\text{type} = \text{Rollback}$





$$\begin{aligned}
& \begin{aligned}
& [path \mapsto path, \\
& \quad value \mapsto Nil, \\
& \quad index \mapsto 0, \\
& \quad deleted \mapsto FALSE]]]] \\
\wedge target = [t \in \text{DOMAIN } Target \mapsto \\
& \quad [path \in \{\} \mapsto \\
& \quad \quad [value \mapsto Nil]]] \\
\wedge mastership = [t \in \text{DOMAIN } Target \mapsto [master \mapsto Nil, term \mapsto 0]] \\
Next \triangleq & \\
& \vee \exists c \in \text{ValidChanges} : \\
& \quad RequestChange(c) \\
& \vee \exists t \in \text{DOMAIN } transaction : \\
& \quad RequestRollback(t) \\
& \vee \exists n \in \text{Node} : \\
& \quad \exists t \in \text{DOMAIN } Target : \\
& \quad \quad SetMaster(n, t) \\
& \quad \vee \exists t \in \text{DOMAIN } Target : \\
& \quad \quad UnsetMaster(t) \\
& \vee \exists n \in \text{Node} : \\
& \quad \exists t \in \text{DOMAIN } transaction : \\
& \quad \quad ReconcileTransaction(n, t) \\
& \vee \exists n \in \text{Node} : \\
& \quad \exists t \in \text{DOMAIN } proposal : \\
& \quad \quad \exists i \in \text{DOMAIN } proposal[t] : \\
& \quad \quad \quad ReconcileProposal(n, t, i) \\
& \vee \exists n \in \text{Node} : \\
& \quad \exists c \in \text{DOMAIN } configuration : \\
& \quad \quad ReconcileConfiguration(n, c) \\
Spec \triangleq & Init \wedge \Box [Next]_{vars} \\
Order \triangleq & \\
& \wedge \forall i, j \in \text{DOMAIN } transaction : \\
& \quad \vee j \leq i \\
& \quad \vee Phase[transaction[i].phase] \geq Phase[transaction[j].phase] \\
& \quad \vee transaction[i].targets \cap transaction[j].targets = \{\} \\
& \quad \vee transaction[j].status \in \{Pending, Failed\} \\
& \wedge \forall t \in \text{DOMAIN } proposal : \\
& \quad \forall i, j \in \text{DOMAIN } proposal[t] : \\
& \quad \quad \vee j \leq i \\
& \quad \quad \vee Phase[proposal[t][i].phase] \geq Phase[proposal[t][j].phase] \\
& \quad \quad \vee proposal[t][j].status \in \{Pending, Failed\} \\
Consistency \triangleq & \\
& \forall t \in \text{DOMAIN } target :
\end{aligned}$$

LET

Compute the transaction indexes that have been applied to the target

$$\begin{aligned} appliedIndexes &\triangleq \{i \in \text{DOMAIN } transaction : \\ &\quad \wedge transaction[i].type = Change \\ &\quad \wedge i \in \text{DOMAIN } proposal[t] \\ &\quad \wedge proposal[t][i].phase = Apply \\ &\quad \wedge proposal[t][i].status = Complete \\ &\quad \wedge t \in \text{DOMAIN } transaction[i].values \\ &\quad \wedge \neg \exists j \in \text{DOMAIN } transaction : \\ &\quad \quad \wedge j > i \\ &\quad \quad \wedge transaction[j].type = Rollback \\ &\quad \quad \wedge transaction[j].rollback = i \\ &\quad \quad \wedge transaction[j].phase = Apply \\ &\quad \quad \wedge transaction[j].status = Complete\} \end{aligned}$$

Compute the set of paths in the target that have been updated by transactions

$$appliedPaths \triangleq \text{UNION } \{ \text{DOMAIN } transaction[i].values[t] : i \in appliedIndexes \}$$

Compute the highest index applied to the target for each path

$$\begin{aligned} pathIndexes &\triangleq [p \in appliedPaths \mapsto \text{CHOOSE } i \in appliedIndexes : \\ &\quad \forall j \in appliedIndexes : \\ &\quad \quad \wedge i \geq j \\ &\quad \quad \wedge p \in \text{DOMAIN } transaction[i].values[t]] \end{aligned}$$

Compute the expected target configuration based on the last indexes applied to the target for each path.

$$expectedConfig \triangleq [p \in \text{DOMAIN } pathIndexes \mapsto transaction[pathIndexes[p]].values[t][p]]$$

IN

$$target[t] = expectedConfig$$

Isolation  $\triangleq$

$$\begin{aligned} &\forall i, j \in \text{DOMAIN } transaction : \\ &\quad \vee j \leq i \\ &\quad \vee transaction[i].targets \cap transaction[j].targets = \{\} \\ &\quad \vee transaction[i].isolation \neq Serializable \\ &\quad \vee \wedge \vee \wedge transaction[i].phase \in \{Commit, Abort\} \\ &\quad \quad \wedge transaction[i].status \in \{Complete, Failed\} \\ &\quad \quad \vee Phase[transaction[i].phase] > Phase[Commit] \\ &\quad \quad \vee Phase[transaction[j].phase] < Phase[Commit] \\ &\quad \wedge \vee \wedge transaction[i].phase \in \{Apply, Abort\} \\ &\quad \quad \wedge transaction[i].status \in \{Complete, Failed\} \\ &\quad \quad \vee Phase[transaction[j].phase] < Phase[Apply] \\ &\quad \vee transaction[j].status = Failed \end{aligned}$$

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

Completion  $\triangleq$

$$\begin{aligned} &\wedge \forall i \in \text{DOMAIN } transaction : \\ &\quad \wedge transaction[i].phase = Commit \end{aligned}$$



$$\begin{aligned}
& \wedge \text{transaction}[i].\text{status} \in \{\text{Complete}, \text{Failed}\} \\
& \wedge \forall i \in \text{DOMAIN } \text{transaction} : \\
& \quad \wedge \text{transaction}[i].\text{phase} = \text{Apply} \\
& \quad \wedge \text{transaction}[i].\text{status} \in \{\text{Complete}, \text{Failed}\} \\
& \wedge \forall t \in \text{DOMAIN } \text{proposal} : \\
& \quad \forall i \in \text{DOMAIN } \text{proposal}[t] : \\
& \quad \quad \wedge \text{proposal}[t][i].\text{phase} = \text{Commit} \\
& \quad \quad \wedge \text{proposal}[t][i].\text{status} \in \{\text{Complete}, \text{Failed}\} \\
& \wedge \forall t \in \text{DOMAIN } \text{proposal} : \\
& \quad \forall i \in \text{DOMAIN } \text{proposal}[t] : \\
& \quad \quad \wedge \text{proposal}[t][i].\text{phase} = \text{Apply} \\
& \quad \quad \wedge \text{proposal}[t][i].\text{status} \in \{\text{Complete}, \text{Failed}\}
\end{aligned}$$

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

---

Type assumptions.

ASSUME *Nil*  $\in$  STRING

ASSUME  $\forall \text{phase} \in \text{DOMAIN } \text{Phase} : \text{phase} \in \text{STRING}$

ASSUME  $\forall \text{status} \in \text{DOMAIN } \text{Status} : \text{status} \in \text{STRING}$

ASSUME  $\wedge \text{IsFiniteSet}(\text{Node})$

$$\begin{aligned}
& \wedge \forall n \in \text{Node} : \\
& \quad \wedge n \notin \text{DOMAIN } \text{Target} \\
& \quad \wedge n \in \text{STRING}
\end{aligned}$$

ASSUME  $\wedge \forall t \in \text{DOMAIN } \text{Target} :$

$$\begin{aligned}
& \quad \wedge t \notin \text{Node} \\
& \quad \wedge t \in \text{STRING} \\
& \quad \wedge \text{Target}[t].\text{persistent} \in \text{BOOLEAN} \\
& \quad \wedge \forall p \in \text{DOMAIN } \text{Target}[t].\text{values} : \\
& \quad \quad \text{IsFiniteSet}(\text{Target}[t].\text{values}[p])
\end{aligned}$$


---

\ \* Modification History  
\ \* Last modified *Mon Feb 07 13:58:48 PST 2022* by *jordanhalterman*  
\ \* Created *Wed Sep 22 13:22:32 PDT 2021* by *jordanhalterman*