
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

$\{$ *Initialize*,
Validate,
Abort,
Commit,
Apply $\}$

Status constants

CONSTANTS

InProgress,
Complete,
Failed

State \triangleq

$\{$ *InProgress*,
Complete,
 $\}$

Failed}

State constants

CONSTANTS

Pending,
Validated,
Committed,
Applied,
Aborted

Status \triangleq
 $\{$ *Pending*,
Validated,
Committed,
Applied,
Aborted $\}$

CONSTANTS

Valid,
Invalid

CONSTANTS

Success,
Failure

The set of all nodes

CONSTANT *Node*

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

Example:

Target \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 ∈
{ Initialize,
  Validate,
  Abort,
  Commit,
  Apply }

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

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

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

TYPE Transaction  $\triangleq$ 
[ type      ::= type ∈ Type,
  isolation ::= isolation ∈ Isolation
  change ::=
    [ target ∈ SUBSET (DOMAIN Target)  $\mapsto$ 
      [ path ∈ SUBSET (DOMAIN Target[target].values)  $\mapsto$ 
        [ value ::= value ∈ STRING,
          delete ::= delete ∈ BOOLEAN ] ] ],
  rollback ::= index ∈ Nat,
  targets ::= targets ∈ SUBSET (DOMAIN Target)
  phase    ::= phase ∈ Phase,
  state    ::= state ∈ State,
  status   ::= status ∈ Status ]

TYPE Proposal  $\triangleq$ 
[ type      ::= type ∈ Type,
  change    ::=
    [ index ::= index ∈ Nat,
      values ::=
        [ path ∈ SUBSET (DOMAIN Target[target].values)  $\mapsto$ 
          [ value ::= value ∈ STRING,
            delete ::= delete ∈ BOOLEAN ] ] ],
  rollback ::=
    [ index ::= index ∈ Nat,
      values ::=
        [ path ∈ SUBSET (DOMAIN Target[target].values)  $\mapsto$ 
          [ value ::= value ∈ STRING,
```

```

    delete ::= delete ∈ BOOLEAN ]]],
    dependency ::= [index ∈ Nat],
    phase ::= phase ∈ Phase,
    state ::= state ∈ State]
TYPE Configuration ≜
[config ::=
  [index ::= index ∈ Nat,
   term ::= term ∈ Nat,
   values ::=
    [path ∈ SUBSET (DOMAIN Target[target]) ↦
     [value ::= value ∈ STRING,
      index ::= index ∈ Nat,
      deleted ::= delete ∈ BOOLEAN ]]],
  proposal ::= [index ::= index ∈ Nat],
  commit ::= [index ::= index ∈ Nat],
  target ::=
    [index ::= index ∈ Nat,
     term ::= term ∈ Nat,
     values ::=
      [path ∈ SUBSET (DOMAIN Target[target]) ↦
       [value ::= value ∈ STRING,
        index ::= index ∈ Nat,
        deleted ::= delete ∈ BOOLEAN ]]],
  state ::= state ∈ State]

```

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

VARIABLE *transaction*

A record of per-target proposals

VARIABLE *proposal*

A record of per-target configurations

VARIABLE *configuration*

A record of target states

VARIABLE *target*

A record of target 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*

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

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

$$\begin{aligned}
\text{Value}(s, t, p) &\triangleq \\
&\text{LET } \text{value} \triangleq \text{CHOOSE } v \in s : v.\text{target} = t \wedge v.\text{path} = p \\
&\text{IN} \\
&\quad [value \mapsto \text{value.value}, \\
&\quad \quad \text{delete} \mapsto \text{value.delete}] \\
\\
\text{Paths}(s, t) &\triangleq \\
&\quad [p \in \{v.\text{path} : v \in \{v \in s : v.\text{target} = t\}\} \mapsto \text{Value}(s, t, p)] \\
\\
\text{Changes}(s) &\triangleq \\
&\quad [t \in \{v.\text{target} : v \in s\} \mapsto \text{Paths}(s, t)] \\
\\
\text{ValidValues}(t, p) &\triangleq \\
&\quad \text{UNION } \{[value \mapsto v, delete \mapsto \text{FALSE}] : v \in \text{Target}[t].\text{values}[p]\}, \{[value \mapsto \text{Nil}, delete \mapsto \text{TRUE}]\} \\
\\
\text{ValidPaths}(t) &\triangleq \\
&\quad \text{UNION } \{[v @@@ [path \mapsto p] : v \in \text{ValidValues}(t, p)] : p \in \text{DOMAIN } \text{Target}[t].\text{values}\} \\
\\
\text{ValidTargets} &\triangleq \\
&\quad \text{UNION } \{[p @@@ [target \mapsto t] : p \in \text{ValidPaths}(t)] : t \in \text{DOMAIN } \text{Target}\}
\end{aligned}$$

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.

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

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.

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

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

$$\begin{aligned}
RequestChange(c) &\triangleq \\
&\wedge \exists isolation \in \{ReadCommitted, Serializable\} : \\
&\quad \wedge transaction' = transaction @@ (NextIndex :> [type \mapsto Change, \\
&\quad \quad \quad isolation \mapsto isolation, \\
&\quad \quad \quad change \mapsto c, \\
&\quad \quad \quad targets \mapsto \{\}, \\
&\quad \quad \quad phase \mapsto Initialize, \\
&\quad \quad \quad state \mapsto InProgress, \\
&\quad \quad \quad status \mapsto Pending]) \\
&\quad \wedge \text{UNCHANGED } \langle proposal, configuration, mastership, target \rangle
\end{aligned}$$

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

$$\begin{aligned}
RequestRollback(i) &\triangleq \\
&\wedge \exists isolation \in \{ReadCommitted, Serializable\} : \\
&\quad \wedge transaction' = transaction @@ (NextIndex :> [type \mapsto Rollback, \\
&\quad \quad \quad isolation \mapsto isolation, \\
&\quad \quad \quad rollback \mapsto i, \\
&\quad \quad \quad targets \mapsto \{\}, \\
&\quad \quad \quad phase \mapsto Initialize, \\
&\quad \quad \quad state \mapsto InProgress, \\
&\quad \quad \quad status \mapsto Pending]) \\
&\quad \wedge \text{UNCHANGED } \langle proposal, configuration, mastership, target \rangle
\end{aligned}$$

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

$$ReconcileTransaction(n, i) \triangleq$$

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{aligned}
& \wedge \vee \wedge \text{transaction}[i].\text{phase} = \text{Initialize} \\
& \wedge \vee \wedge \text{transaction}[i].\text{state} = \text{InProgress} \\
& \quad \text{All prior transaction must be initialized before proceeding} \\
& \quad \text{to initialize this transaction.} \\
& \wedge \neg \exists j \in \text{DOMAIN } \text{transaction} : \\
& \quad \wedge j < i \\
& \quad \wedge \text{transaction}[j].\text{phase} = \text{Initialize} \\
& \quad \wedge \text{transaction}[j].\text{state} = \text{InProgress} \\
& \quad \text{If the transaction's targets are not yet set, create proposals} \\
& \quad \text{and add targets to the transaction state.} \\
& \wedge \vee \wedge \text{transaction}[i].\text{targets} = \{\} \\
& \quad \text{If the transaction is a change, the targets are taken} \\
& \quad \text{from the change values.} \\
& \wedge \vee \wedge \text{transaction}[i].\text{type} = \text{Change} \\
& \quad \wedge \text{transaction}' = [\text{transaction EXCEPT } ![i].\text{targets} = \text{DOMAIN } \text{transaction}[i].\text{change}] \\
& \quad \wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto \\
& \quad \quad \text{IF } t \in \text{DOMAIN } \text{transaction}[i].\text{change} \text{ THEN} \\
& \quad \quad \quad \text{proposal}[t] @ @ (i :> [type \mapsto \text{Change},} \\
& \quad \quad \quad \quad \text{change} \mapsto \\
& \quad \quad \quad \quad \quad [index \mapsto i, \\
& \quad \quad \quad \quad \quad \text{values} \mapsto \text{transaction}[i].\text{change}[t]], \\
& \quad \quad \quad \text{rollback} \mapsto \\
& \quad \quad \quad \quad [index \mapsto 0, \\
& \quad \quad \quad \quad \text{values} \mapsto \text{Nil}], \\
& \quad \quad \quad \text{dependency} \mapsto [index \mapsto 0], \\
& \quad \quad \quad \text{phase} \mapsto \text{Initialize}, \\
& \quad \quad \quad \text{state} \mapsto \text{InProgress}]) \\
& \quad \quad \text{ELSE} \\
& \quad \quad \quad \text{proposal}[t] \\
& \quad \quad \text{If the transaction is a rollback, the targets affected are} \\
& \quad \quad \text{the targets of the change transaction being rolled back.} \\
& \vee \wedge \text{transaction}[i].\text{type} = \text{Rollback} \\
& \quad \text{If the rollback index is a valid Change transaction,} \\
& \quad \text{initialize proposals for all of the Change targets.} \\
& \wedge \vee \wedge \text{transaction}[i].\text{rollback} \in \text{DOMAIN } \text{transaction} \\
& \quad \wedge \text{transaction}[\text{transaction}[i].\text{rollback}].\text{type} = \text{Change} \\
& \quad \wedge \text{transaction}' = [\text{transaction EXCEPT } ![i].\text{targets} = \\
& \quad \quad \text{DOMAIN } \text{transaction}[\text{transaction}[i].\text{rollback}].\text{change}] \\
& \quad \wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto
\end{aligned}$$

IF $t \in \text{DOMAIN } \text{transaction}[\text{transaction}[i].\text{rollback}].\text{change}$ THEN
 $\text{proposal}[t] @@ (i :> [\text{type} \mapsto \text{Rollback},$
 $\text{change} \mapsto$
 $\text{index} \mapsto 0,$
 $\text{values} \mapsto \text{Nil}],$
 $\text{rollback} \mapsto$
 $\text{index} \mapsto \text{transaction}[i].\text{rollback},$
 $\text{values} \mapsto \text{Nil}],$
 $\text{dependency} \mapsto [\text{index} \mapsto 0],$
 $\text{phase} \mapsto \text{Initialize},$
 $\text{state} \mapsto \text{InProgress}])$
 ELSE
 $\text{proposal}[t]$
 If the rollback index is not a valid *Change* transaction
 fail the *Rollback* transaction.
 $\vee \wedge \vee \wedge \text{transaction}[i].\text{rollback} \in \text{DOMAIN } \text{transaction}$
 $\wedge \text{transaction}[\text{transaction}[i].\text{rollback}].\text{type} = \text{Rollback}$
 $\vee \text{transaction}[i].\text{rollback} \notin \text{DOMAIN } \text{transaction}$
 $\wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{state} = \text{Failed}]$
 $\wedge \text{UNCHANGED } \langle \text{proposal} \rangle$
 If the transaction's proposals have been initialized, check proposals
 for completion or failures.
 $\vee \wedge \text{transaction}[i].\text{targets} \neq \{\}$
 If all proposals have been *Complete*, mark the transaction *Complete*.
 $\wedge \vee \wedge \forall t \in \text{transaction}[i].\text{targets} :$
 $\wedge \text{proposal}[t][i].\text{phase} = \text{Initialize}$
 $\wedge \text{proposal}[t][i].\text{state} = \text{Complete}$
 $\wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{state} = \text{Complete}]$
 $\wedge \text{UNCHANGED } \langle \text{proposal} \rangle$
 If any proposal has been *Failed*, mark the transaction *Failed*.
 $\vee \wedge \exists t \in \text{transaction}[i].\text{targets} :$
 $\wedge \text{proposal}[t][i].\text{phase} = \text{Initialize}$
 $\wedge \text{proposal}[t][i].\text{state} = \text{Failed}$
 $\wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{state} = \text{Failed}]$
 $\wedge \text{UNCHANGED } \langle \text{proposal} \rangle$
 Once the transaction has been *Initialized*, proceed to the *Validate* phase.
 If any of the transaction's proposals depend on a *Serializable* transaction,
 verify the dependency has been *Validated* to preserve serializability before
 moving the transaction to the *Validate* phase.
 $\vee \wedge \text{transaction}[i].\text{state} = \text{Complete}$
 $\wedge \forall t \in \text{transaction}[i].\text{targets} :$
 $\wedge \text{proposal}[t][i].\text{dependency.index} \in \text{DOMAIN } \text{transaction}$
 $\wedge \text{transaction}[\text{proposal}[t][i].\text{dependency.index}].\text{isolation} = \text{Serializable}$
 $\Rightarrow \text{transaction}[\text{proposal}[t][i].\text{dependency.index}].\text{status} \in \{ \text{Validated}, \text{Committed}, \text{Applied}, \text{A} \}$
 $\wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{phase} = \text{Validate},$

$![i].state = InProgress]$

\wedge UNCHANGED $\langle proposal \rangle$

If the transaction failed initialization, proceed to the *Abort* phase to ensure indexes are still updated for the target configurations.

$\vee \wedge transaction[i].state = Failed$
 $\wedge transaction' = [transaction \text{ EXCEPT } ![i].phase = Abort,$
 $![i].state = InProgress]$

\wedge UNCHANGED $\langle proposal \rangle$

$\vee \wedge transaction[i].phase = Validate$
 $\wedge \vee \wedge transaction[i].state = InProgress$

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

$\wedge \vee \wedge \exists t \in transaction[i].targets :$
 $\wedge proposal[t][i].phase \neq Validate$
 $\wedge proposal' = [proposal \text{ EXCEPT } ![t] =$
 $[proposal[t] \text{ EXCEPT } ![i].phase = Validate,$
 $![i].state = InProgress]]$

\wedge UNCHANGED $\langle transaction \rangle$

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

$\vee \wedge \forall t \in transaction[i].targets :$
 $\wedge proposal[t][i].phase = Validate$
 $\wedge proposal[t][i].state = Complete$
 $\wedge transaction' = [transaction \text{ EXCEPT } ![i].state = Complete,$
 $![i].status = Validated]$

\wedge UNCHANGED $\langle proposal \rangle$

If any proposal has been *Failed*, mark the transaction *Failed*.

$\vee \wedge \exists t \in transaction[i].targets :$
 $\wedge proposal[t][i].phase = Validate$
 $\wedge proposal[t][i].state = Failed$
 $\wedge transaction' = [transaction \text{ EXCEPT } ![i].state = Failed]$
 \wedge 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.

$\vee \wedge transaction[i].state = Complete$
 $\wedge \forall t \in transaction[i].targets :$
 $\wedge proposal[t][i].dependency.index \in \text{DOMAIN } transaction$
 $\wedge transaction[proposal[t][i].dependency.index].isolation = Serializable$
 $\Rightarrow transaction[proposal[t][i].dependency.index].status \in \{Committed, Applied, Aborted\}$
 $\wedge transaction' = [transaction \text{ EXCEPT } ![i].phase = Commit,$
 $![i].state = InProgress]$

\wedge UNCHANGED $\langle proposal \rangle$

If the transaction failed validation, proceed to the *Abort* phase to ensure indexes are still updated for the target configurations.

$\vee \wedge transaction[i].state = Failed$

$$\begin{aligned}
& \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{phase} = \text{Abort}, \\
& \hspace{15em} ![i].\text{state} = \text{InProgress}] \\
& \wedge \text{UNCHANGED } \langle \text{proposal} \rangle \\
\vee & \wedge \text{transaction}[i].\text{phase} = \text{Commit} \\
& \wedge \vee \wedge \text{transaction}[i].\text{state} = \text{InProgress} \\
& \quad \text{Move the transaction's proposals to the } \textit{Committing} \text{ state} \\
& \wedge \vee \wedge \exists t \in \text{transaction}[i].\text{targets} : \\
& \quad \wedge \text{proposal}[t][i].\text{phase} \neq \text{Commit} \\
& \quad \wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = \\
& \hspace{15em} [\text{proposal}[t] \text{ EXCEPT } ![i].\text{phase} = \text{Commit}, \\
& \hspace{15em} ![i].\text{state} = \text{InProgress}]] \\
& \wedge \text{UNCHANGED } \langle \text{transaction} \rangle \\
& \quad \text{If all proposals have been } \textit{Complete}, \text{ mark the transaction } \textit{Complete}. \\
& \vee \wedge \forall t \in \text{transaction}[i].\text{targets} : \\
& \quad \wedge \text{proposal}[t][i].\text{phase} = \text{Commit} \\
& \quad \wedge \text{proposal}[t][i].\text{state} = \text{Complete} \\
& \quad \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{state} = \text{Complete}, \\
& \hspace{15em} ![i].\text{status} = \text{Committed}] \\
& \wedge \text{UNCHANGED } \langle \text{proposal} \rangle \\
& \quad \text{Once the transaction has been } \textit{Committed}, \text{ proceed to the } \textit{Apply} \text{ phase.} \\
& \quad \text{If any of the transaction's proposals depend on a } \textit{Serializable} \text{ transaction,} \\
& \quad \text{verify the dependency has been } \textit{Applied} \text{ to preserve serializability before} \\
& \quad \text{moving the transaction to the } \textit{Apply} \text{ phase.} \\
& \vee \wedge \text{transaction}[i].\text{state} = \text{Complete} \\
& \quad \wedge \forall t \in \text{transaction}[i].\text{targets} : \\
& \quad \quad \wedge \text{proposal}[t][i].\text{dependency.index} \in \text{DOMAIN } \text{transaction} \\
& \quad \quad \wedge \text{transaction}[\text{proposal}[t][i].\text{dependency.index}].\text{isolation} = \textit{Serializable} \\
& \quad \quad \Rightarrow \text{transaction}[\text{proposal}[t][i].\text{dependency.index}].\text{status} \in \{\textit{Applied}, \textit{Aborted}\} \\
& \quad \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{phase} = \textit{Apply}, \\
& \hspace{15em} ![i].\text{state} = \textit{InProgress}] \\
& \wedge \text{UNCHANGED } \langle \text{proposal} \rangle \\
\vee & \wedge \text{transaction}[i].\text{phase} = \textit{Apply} \\
& \wedge \text{transaction}[i].\text{state} = \textit{InProgress} \\
& \quad \text{Move the transaction's proposals to the } \textit{Applying} \text{ state} \\
& \wedge \vee \wedge \exists t \in \text{transaction}[i].\text{targets} : \\
& \quad \wedge \text{proposal}[t][i].\text{phase} \neq \textit{Apply} \\
& \quad \wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = \\
& \hspace{15em} [\text{proposal}[t] \text{ EXCEPT } ![i].\text{phase} = \textit{Apply}, \\
& \hspace{15em} ![i].\text{state} = \textit{InProgress}]] \\
& \wedge \text{UNCHANGED } \langle \text{transaction} \rangle \\
& \quad \text{If all proposals have been } \textit{Complete}, \text{ mark the transaction } \textit{Complete}. \\
& \vee \wedge \forall t \in \text{transaction}[i].\text{targets} : \\
& \quad \wedge \text{proposal}[t][i].\text{phase} = \textit{Apply} \\
& \quad \wedge \text{proposal}[t][i].\text{state} = \textit{Complete} \\
& \quad \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{state} = \textit{Complete},
\end{aligned}$$

$![i].status = Applied]$

\wedge UNCHANGED $\langle proposal \rangle$

If any proposal has been *Failed*, mark the transaction *Failed*.

$\vee \wedge \exists t \in transaction[i].targets :$

$\wedge proposal[t][i].phase = Apply$

$\wedge proposal[t][i].state = Failed$

$\wedge transaction' = [transaction \text{ EXCEPT } ![i].state = Failed]$

\wedge UNCHANGED $\langle proposal \rangle$

The *Aborting* state is used to clean up transactions that have failed during the *Initializing* or *Validating* phases.

$\vee \wedge transaction[i].phase = Abort$

$\wedge transaction[i].state = InProgress$

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

$\wedge \vee \wedge \exists t \in transaction[i].targets :$

$\wedge proposal[t][i].phase \neq Abort$

$\wedge proposal' = [proposal \text{ EXCEPT } ![t] =$

$[proposal[t] \text{ EXCEPT } ![i].phase = Abort,$

$![i].state = InProgress]]$

\wedge UNCHANGED $\langle transaction \rangle$

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

$\vee \wedge \forall t \in transaction[i].targets :$

$\wedge proposal[t][i].phase = Abort$

$\wedge proposal[t][i].state = Complete$

$\wedge transaction' = [transaction \text{ EXCEPT } ![i].state = Complete,$

$![i].status = Aborted]$

\wedge UNCHANGED $\langle proposal \rangle$

\wedge UNCHANGED $\langle configuration, mastership, target \rangle$

Reconcile a proposal

$ReconcileProposal(n, t, i) \triangleq$

$\wedge \vee \wedge proposal[t][i].phase = Initialize$

$\wedge proposal[t][i].state = InProgress$

$\wedge proposal' = [proposal \text{ EXCEPT } ![t] =$

$[proposal[t] \text{ EXCEPT } ![i].state = Complete,$

$![i].dependency.index = configuration[t].proposal.index]]$

$\wedge configuration' = [configuration \text{ EXCEPT } ![t].proposal.index = i]$

\wedge 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.

$\vee \wedge proposal[t][i].phase = Validate$

$\wedge proposal[t][i].state = InProgress$

$\wedge configuration[t].commit.index = proposal[t][i].dependency.index$

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

$\wedge \vee \wedge proposal[t][i].type = Change$

$$\begin{aligned} \wedge \text{ LET } \textit{rollbackIndex} &\triangleq \textit{configuration}[t].\textit{config.index} \\ \textit{rollbackValues} &\triangleq [p \in \text{DOMAIN } \textit{proposal}[t][i].\textit{change.values} \mapsto \\ &\quad \text{IF } p \in \text{DOMAIN } \textit{configuration}[t].\textit{config.values} \text{ THEN} \\ &\quad \quad \textit{configuration}[t].\textit{config.values}[p] \\ &\quad \text{ELSE} \\ &\quad \quad [value \mapsto Nil, \\ &\quad \quad \quad delete \mapsto \text{TRUE}]] \end{aligned}$$
$$\text{IN} \quad \exists r \in \{Valid, Invalid\} :$$
$$\vee \wedge r = \textit{Valid}$$
$$\vee \wedge r = Invalid$$

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

$$\wedge \vee \wedge \text{proposal}[t][\text{proposal}[t][i].\text{rollback.index}].\text{type} = \text{Change}$$
$$\wedge \vee \wedge configuration[t].config.index = proposal[t][i].rollback.index$$
$$\text{IN} \quad \exists r \in \{Valid, Invalid\} :$$
$$\vee \wedge r = \textit{Valid}$$
$$\vee \wedge r = Invalid$$

If the *Rollback* target is not the most recent change to the configuration,

fail validation for the proposal.

$$\vee \wedge \text{configuration}[t].\text{config.index} \neq \text{proposal}[t][i].\text{rollback.index}$$

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

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

$$\vee \wedge \text{proposal}[t][\text{proposal}[t][i].\text{rollback.index}].\text{type} = \text{Rollback}$$

$$\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] =$$

$$[\text{proposal}[t] \text{ EXCEPT } ![i].\text{state} = \text{Failed}]]$$

$$\wedge \text{UNCHANGED } \langle \text{configuration}, \text{target} \rangle$$

While in the *Commit* state, commit the proposed changes to the configuration.

$$\vee \wedge \text{proposal}[t][i].\text{phase} = \text{Commit}$$

$$\wedge \text{proposal}[t][i].\text{state} = \text{InProgress}$$

Only commit the proposal if the prior proposal has already been committed.

$$\wedge \text{configuration}[t].\text{commit.index} = \text{proposal}[t][i].\text{dependency.index}$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![t].\text{config.values} = \text{proposal}[t][i].\text{change.values},$$

$$![t].\text{config.index} = \text{proposal}[t][i].\text{change.index},$$

$$![t].\text{commit.index} = i]$$

$$\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [\text{proposal}[t] \text{ EXCEPT } ![i].\text{state} = \text{Complete}]]$$

$$\wedge \text{UNCHANGED } \langle \text{target} \rangle$$

While in the *Apply* phase, apply the proposed changes to the target.

$$\vee \wedge \text{proposal}[t][i].\text{phase} = \text{Apply}$$

$$\wedge \text{proposal}[t][i].\text{state} = \text{InProgress}$$

$$\wedge \text{configuration}[t].\text{target.index} = \text{proposal}[t][i].\text{dependency.index}$$

$$\wedge \text{configuration}[t].\text{target.term} = \text{mastership}[t].\text{term}$$

$$\wedge \text{mastership}[t].\text{master} = n$$

Model successful and failed target update requests.

$$\wedge \exists r \in \{\text{Success}, \text{Failure}\} :$$

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

$$\wedge \text{target}' = [\text{target} \text{ EXCEPT } ![t] = \text{proposal}[t][i].\text{change.values} @@ \text{target}[t]]$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT }$$

$$![t].\text{target.index} = i,$$

$$![t].\text{target.values} = \text{proposal}[t][i].\text{change.values}$$

$$@@ \text{configuration}[t].\text{target.values}]$$

$$\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [\text{proposal}[t] \text{ EXCEPT } ![i].\text{state} = \text{Complete}]]$$

If the proposal could not be applied, update the configuration's applied index
and mark the proposal *Failed*.

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

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![t].\text{target.index} = i]$$

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

$$\wedge \text{UNCHANGED } \langle \text{target} \rangle$$

$$\vee \wedge \text{proposal}[t][i].\text{phase} = \text{Abort}$$

$$\wedge \text{proposal}[t][i].\text{state} = \text{InProgress}$$

The *commit.index* will always be greater than or equal to the *target.index*.
If only the *commit.index* matches the proposal's *dependency.index*, update
the *commit.index* to enable commits of later proposals, but do not

mark the *Abort* phase *Complete* until the *target.index* has been incremented.

$$\wedge \vee \wedge \text{configuration}[t].\text{commit.index} = \text{proposal}[t][i].\text{dependency.index}$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![t].\text{commit.index} = i]$$

$$\wedge \text{UNCHANGED } \langle \text{proposal} \rangle$$

If the configuration's *target.index* matches the proposal's *dependency.index*,
update the *target.index* and mark the proposal *Complete* for the *Abort* phase.

$$\vee \wedge \text{configuration}[t].\text{commit.index} \geq i$$

$$\wedge \text{configuration}[t].\text{target.index} = \text{proposal}[t][i].\text{dependency.index}$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![t].\text{target.index} = i]$$

$$\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [\text{proposal}[t] \text{ EXCEPT } ![i].\text{state} = \text{Complete}]]$$

If both the configuration's *commit.index* and *target.index* match the
proposal's *dependency.index*, update the *commit.index* and *target.index*
and mark the proposal *Complete* for the *Abort* phase.

$$\vee \wedge \text{configuration}[t].\text{commit.index} = \text{proposal}[t][i].\text{dependency.index}$$

$$\wedge \text{configuration}[t].\text{target.index} = \text{proposal}[t][i].\text{dependency.index}$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![t].\text{commit.index} = i,$$

$$\phantom{\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } } ![t].\text{target.index} = i]$$

$$\wedge \text{proposal}' = [\text{proposal} \text{ EXCEPT } ![t] = [\text{proposal}[t] \text{ EXCEPT } ![i].\text{state} = \text{Complete}]]$$

$$\wedge \text{UNCHANGED } \langle \text{target} \rangle$$

$$\wedge \text{UNCHANGED } \langle \text{transaction}, \text{mastership} \rangle$$

This section models the Configuration reconciler.

$\text{ReconcileConfiguration}(n, t) \triangleq$

$$\wedge \vee \wedge \text{Target}[t].\text{persistent}$$

$$\wedge \text{configuration}[t].\text{state} \neq \text{Complete}$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![t].\text{state} = \text{Complete}]$$

$$\wedge \text{UNCHANGED } \langle \text{target} \rangle$$

$$\vee \wedge \neg \text{Target}[t].\text{persistent}$$

$$\wedge \vee \text{mastership}[t].\text{term} > \text{configuration}[t].\text{config.term}$$

$$\vee \wedge \text{mastership}[t].\text{term} = \text{configuration}[t].\text{config.term}$$

$$\wedge \text{mastership}[t].\text{master} = \text{Nil}$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![t].\text{config.term} = \text{mastership}[t].\text{term},$$

$$\phantom{\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } } ![t].\text{state} = \text{InProgress}]$$

$$\wedge \text{UNCHANGED } \langle \text{target} \rangle$$

$$\vee \wedge \text{configuration}[t].\text{state} = \text{InProgress}$$

$$\wedge \text{mastership}[t].\text{term} = \text{configuration}[t].\text{config.term}$$

$$\wedge \text{mastership}[t].\text{master} = n$$

$$\wedge \text{target}' = [\text{target} \text{ EXCEPT } ![t] = \text{configuration}[t].\text{target.values}]$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![t].\text{target.term} = \text{mastership}[t].\text{term},$$

$$\phantom{\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } } ![t].\text{state} = \text{Complete}]$$

$$\wedge \text{UNCHANGED } \langle \text{proposal}, \text{transaction}, \text{mastership} \rangle$$

Formal specification, constraints, and theorems.

$Init \triangleq$

$$\begin{aligned}
& \wedge transaction = [i \in \{\} \mapsto \\
& \quad [type \mapsto Change, \\
& \quad \quad phase \mapsto Initialize, \\
& \quad \quad state \mapsto InProgress, \\
& \quad \quad status \mapsto Pending]] \\
& \wedge proposal = [t \in \text{DOMAIN } Target \mapsto \\
& \quad [i \in \{\} \mapsto \\
& \quad \quad [phase \mapsto Initialize, \\
& \quad \quad \quad state \mapsto InProgress]]] \\
& \wedge configuration = [t \in \text{DOMAIN } Target \mapsto \\
& \quad [state \mapsto InProgress, \\
& \quad \quad config \mapsto \\
& \quad \quad \quad [index \mapsto 0, \\
& \quad \quad \quad \quad term \mapsto 0, \\
& \quad \quad \quad \quad values \mapsto \\
& \quad \quad \quad \quad \quad [path \in \{\} \mapsto \\
& \quad \quad \quad \quad \quad \quad [path \mapsto path, \\
& \quad \quad \quad \quad \quad \quad \quad value \mapsto Nil, \\
& \quad \quad \quad \quad \quad \quad \quad index \mapsto 0, \\
& \quad \quad \quad \quad \quad \quad \quad deleted \mapsto FALSE]]], \\
& \quad \quad \quad proposal \mapsto [index \mapsto 0], \\
& \quad \quad \quad commit \mapsto [index \mapsto 0], \\
& \quad \quad \quad target \mapsto \\
& \quad \quad \quad \quad [index \mapsto 0, \\
& \quad \quad \quad \quad \quad term \mapsto 0, \\
& \quad \quad \quad \quad \quad values \mapsto \\
& \quad \quad \quad \quad \quad \quad [path \in \{\} \mapsto \\
& \quad \quad \quad \quad \quad \quad \quad [path \mapsto path, \\
& \quad \quad \quad \quad \quad \quad \quad \quad value \mapsto Nil, \\
& \quad \quad \quad \quad \quad \quad \quad \quad index \mapsto 0, \\
& \quad \quad \quad \quad \quad \quad \quad \quad deleted \mapsto FALSE]]]]] \\
& \wedge target = [t \in \text{DOMAIN } Target \mapsto \\
& \quad [path \in \{\} \mapsto \\
& \quad \quad [value \mapsto Nil]]] \\
& \wedge mastership = [t \in \text{DOMAIN } Target \mapsto [master \mapsto Nil, term \mapsto 0]]
\end{aligned}$$

$Next \triangleq$

$$\begin{aligned}
& \forall \exists c \in ValidChanges : \\
& \quad RequestChange(c) \\
& \forall \exists t \in \text{DOMAIN } transaction : \\
& \quad RequestRollback(t) \\
& \forall \exists n \in Node : \\
& \quad \exists t \in \text{DOMAIN } Target :
\end{aligned}$$

$$\begin{aligned}
& \text{SetMaster}(n, t) \\
& \forall \exists t \in \text{DOMAIN } \text{Target} : \\
& \quad \text{UnsetMaster}(t) \\
& \forall \exists n \in \text{Node} : \\
& \quad \exists t \in \text{DOMAIN } \text{transaction} : \\
& \quad \quad \text{ReconcileTransaction}(n, t) \\
& \forall \exists n \in \text{Node} : \\
& \quad \exists t \in \text{DOMAIN } \text{proposal} : \\
& \quad \quad \exists i \in \text{DOMAIN } \text{proposal}[t] : \\
& \quad \quad \quad \text{ReconcileProposal}(n, t, i) \\
& \forall \exists n \in \text{Node} : \\
& \quad \exists c \in \text{DOMAIN } \text{configuration} : \\
& \quad \quad \text{ReconcileConfiguration}(n, c) \\
\text{Spec} & \triangleq \text{Init} \wedge \Box[\text{Next}]_{\text{vars}} \\
\text{Order} & \triangleq \\
& \forall t \in \text{DOMAIN } \text{proposal} : \\
& \quad \forall i \in \text{DOMAIN } \text{proposal}[t] : \\
& \quad \quad \wedge \wedge \text{proposal}[t][i].\text{phase} = \text{Commit} \\
& \quad \quad \wedge \text{proposal}[t][i].\text{state} = \text{InProgress} \\
& \quad \quad \Rightarrow \neg \exists j \in \text{DOMAIN } \text{proposal}[t] : \\
& \quad \quad \quad \wedge j > i \\
& \quad \quad \quad \wedge \text{proposal}[t][j].\text{phase} = \text{Commit} \\
& \quad \quad \quad \wedge \text{proposal}[t][j].\text{state} = \text{Complete} \\
& \quad \wedge \wedge \text{proposal}[t][i].\text{phase} = \text{Apply} \\
& \quad \quad \wedge \text{proposal}[t][i].\text{state} = \text{InProgress} \\
& \quad \quad \Rightarrow \neg \exists j \in \text{DOMAIN } \text{proposal}[t] : \\
& \quad \quad \quad \wedge j > i \\
& \quad \quad \quad \wedge \text{proposal}[t][j].\text{phase} = \text{Apply} \\
& \quad \quad \quad \wedge \text{proposal}[t][j].\text{state} = \text{Complete} \\
\text{Consistency} & \triangleq \\
& \forall t \in \text{DOMAIN } \text{target} : \\
& \quad \text{LET} \\
& \quad \quad \text{Compute the transaction indexes that have been applied to the target} \\
& \quad \text{targetIndexes} \triangleq \{i \in \text{DOMAIN } \text{transaction} : \\
& \quad \quad \quad \wedge i \in \text{DOMAIN } \text{proposal}[t] \\
& \quad \quad \quad \wedge \text{proposal}[t][i].\text{phase} = \text{Apply} \\
& \quad \quad \quad \wedge \text{proposal}[t][i].\text{state} = \text{Complete} \\
& \quad \quad \quad \wedge t \in \text{transaction}[i].\text{targets} \\
& \quad \quad \quad \wedge \neg \exists j \in \text{DOMAIN } \text{transaction} : \\
& \quad \quad \quad \quad \wedge j > i \\
& \quad \quad \quad \quad \wedge \text{transaction}[j].\text{type} = \text{Rollback} \\
& \quad \quad \quad \quad \wedge \text{transaction}[j].\text{rollback} = i \\
& \quad \quad \quad \quad \wedge \text{transaction}[j].\text{phase} = \text{Apply} \\
& \quad \quad \quad \}
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{transaction}[j].\text{state} = \text{Complete}\} \\
& \text{Compute the set of paths in the target that have been updated by transactions} \\
\text{appliedPaths} & \triangleq \text{UNION } \{\text{DOMAIN } \text{proposal}[t][i].\text{change.values} : i \in \text{targetIndexes}\} \\
& \text{Compute the highest index applied to the target for each path} \\
\text{pathIndexes} & \triangleq [p \in \text{appliedPaths} \mapsto \text{CHOOSE } i \in \text{targetIndexes} : \\
& \quad \forall j \in \text{targetIndexes} : \\
& \quad \quad \wedge i \geq j \\
& \quad \quad \wedge p \in \text{DOMAIN } \text{proposal}[t][i].\text{change.values}] \\
& \text{Compute the expected target configuration based on the last indexes applied} \\
& \text{to the target for each path.} \\
\text{expectedConfig} & \triangleq [p \in \text{DOMAIN } \text{pathIndexes} \mapsto \text{proposal}[t][\text{pathIndexes}[p]].\text{change.values}[p]] \\
\text{IN} & \\
& \text{target}[t] = \text{expectedConfig} \\
\text{Isolation} & \triangleq \\
& \forall i \in \text{DOMAIN } \text{transaction} : \\
& \quad \wedge \text{transaction}[i].\text{phase} = \text{Commit} \\
& \quad \wedge \text{transaction}[i].\text{state} = \text{InProgress} \\
& \quad \wedge \text{transaction}[i].\text{isolation} = \text{Serializable} \\
& \quad \Rightarrow \neg \exists j \in \text{DOMAIN } \text{transaction} : \\
& \quad \quad \wedge j > i \\
& \quad \quad \wedge \text{transaction}[j].\text{targets} \cap \text{transaction}[i].\text{targets} \neq \{\} \\
& \quad \quad \wedge \text{transaction}[j].\text{phase} = \text{Commit} \\
& \quad \wedge \text{transaction}[i].\text{phase} = \text{Apply} \\
& \quad \wedge \text{transaction}[i].\text{state} = \text{InProgress} \\
& \quad \wedge \text{transaction}[i].\text{isolation} = \text{Serializable} \\
& \quad \Rightarrow \neg \exists j \in \text{DOMAIN } \text{transaction} : \\
& \quad \quad \wedge j > i \\
& \quad \quad \wedge \text{transaction}[j].\text{targets} \cap \text{transaction}[i].\text{targets} \neq \{\} \\
& \quad \quad \wedge \text{transaction}[j].\text{phase} = \text{Apply} \\
\text{Safety} & \triangleq \Box(\text{Order} \wedge \text{Consistency} \wedge \text{Isolation}) \\
\text{THEOREM } \text{Spec} & \Rightarrow \text{Safety} \\
\text{Completion} & \triangleq \\
& \forall i \in \text{DOMAIN } \text{transaction} : \\
& \quad \wedge \text{transaction}[i].\text{phase} \in \{\text{Apply}, \text{Abort}\} \\
& \quad \wedge \text{transaction}[i].\text{state} = \text{Complete} \\
\text{Liveness} & \triangleq \Diamond \text{Completion} \\
\text{THEOREM } \text{Spec} & \Rightarrow \text{Liveness}
\end{aligned}$$

Type assumptions.

```

ASSUME  $Nil \in \text{STRING}$ 

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

ASSUME  $\forall state \in State : state \in \text{STRING}$ 

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

ASSUME  $\wedge IsFiniteSet(Node)$ 
       $\wedge \forall n \in Node :$ 
         $\wedge n \notin \text{DOMAIN } Target$ 
         $\wedge n \in \text{STRING}$ 

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

```

```

\ * Modification History
\ * Last modified Thu Feb 10 12:45:47 PST 2022 by jordanhalterman
\ * Created Wed Sep 22 13:22:32 PDT 2021 by jordanhalterman

```