
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: $Target \triangleq [$
 $\quad target1 \mapsto [\textit{persistent} \mapsto \text{FALSE}, \textit{values} \mapsto [$
 $\quad \quad path1 \mapsto \{ "value1", "value2" \},$
 $\quad \quad path2 \mapsto \{ "value2", "value3" \}]],$
 $\quad target2 \mapsto [\textit{persistent} \mapsto \text{TRUE}, \textit{values} \mapsto [$
 $\quad \quad path2 \mapsto \{ "value3", "value4" \},$
 $\quad \quad path3 \mapsto \{ "value4", "value5" \}]]]$

CONSTANT *Target*

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

ASSUME *Nil* ∈ STRING

ASSUME *Pending* ∈ STRING

ASSUME *Initializing* ∈ STRING

ASSUME *Initialized* ∈ STRING

ASSUME *Validating* ∈ STRING

ASSUME *Validated* ∈ STRING

ASSUME *Committing* ∈ STRING

ASSUME *Committed* ∈ STRING

ASSUME *Applying* ∈ STRING

ASSUME *Applied* ∈ STRING

ASSUME *Synchronizing* ∈ STRING

ASSUME *Synchronized* ∈ STRING

ASSUME *Persisted* ∈ STRING

ASSUME *Failed* ∈ STRING

ASSUME $\wedge IsFiniteSet(Node)$

$\wedge \forall n \in Node :$

$\wedge n \notin \text{DOMAIN } Target$

$\wedge n \in \text{STRING}$

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

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  $\triangleq$  [
  type      ::= type ∈ Type,
  index     ::= index ∈ Nat,
  isolation ::= isolation ∈ { IsolationDefault, IsolationSerializable }
  values ::= [
    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)
    status ::= status ∈ Status ]

TYPE Proposal  $\triangleq$  [
  type      ::= type ∈ Type,
  index     ::= index ∈ Nat,
  values     ::= [ path ∈ SUBSET (DOMAIN Target[target].values)  $\mapsto$  [
    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 masterhips

VARIABLE *mastership*

vars ≜ ⟨*transaction*, *proposal*, *configuration*, *mastership*, *target*⟩

This section models *mastership* for the configuration service.

Mastership is used primarily to track the lifecycle of individual configuration targets and react to state changes on the southbound. Each target is assigned a master from the *Node* set, and masters can be unset when the target disconnects.

Set node n as the master for target t
 $SetMaster(n, t) \triangleq$
 $\wedge mastership[t].master \neq n$
 $\wedge mastership' = [mastership \text{ EXCEPT } ![t].term = mastership[t].term + 1,$
 $\phantom{\wedge mastership' = [mastership \text{ EXCEPT } } ![t].master = n]$
 $\wedge \text{UNCHANGED } \langle transaction, proposal, configuration, target \rangle$

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

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

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

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

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

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

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

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

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.

$ValidChanges \triangleq$
 $\text{LET } changeSets \triangleq \{ s \in \text{SUBSET } ValidTargets :$
 $\phantom{\text{LET } changeSets \triangleq } \forall t \in \text{DOMAIN } Target : \}$

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

IN
 $\{ \text{Changes}(s) : s \in \text{changeSets} \}$

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

Add a set of changes 'c' to the transaction log
 $\text{RequestChange}(c) \triangleq$
 $\wedge \exists \text{isolation} \in \{ \text{ReadCommitted}, \text{Serializable} \} :$
 $\wedge \text{transaction}' = \text{transaction} @@ (\text{NextIndex} :> [\text{type} \mapsto \text{Change},$
 $\text{index} \mapsto \text{NextIndex},$
 $\text{isolation} \mapsto \text{isolation},$
 $\text{values} \mapsto c,$
 $\text{targets} \mapsto \{ \},$
 $\text{status} \mapsto \text{Initializing}])$
 $\wedge \text{UNCHANGED } \langle \text{proposal}, \text{configuration}, \text{mastership}, \text{target} \rangle$

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

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

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

$ReconcileTransaction(n, i) \triangleq$

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.

$\wedge \vee \wedge transaction[i].status = Initializing$

Serialize transaction initialization

$\wedge i - 1 \in \text{DOMAIN } transaction \Rightarrow$

$Phase(transaction[i - 1].status) > Phase(Initializing)$

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

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

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

$\wedge \vee \wedge transaction[i].type = Change$

$\wedge transaction' = [transaction \text{ EXCEPT } ![i].targets = \text{DOMAIN } transaction[i].values]$

$\wedge proposal' = [t \in \text{DOMAIN } proposal \mapsto$

IF $t \in \text{DOMAIN } transaction[i].values$ THEN

$proposal[t] @@ (i :=$	$[type$	$\mapsto Change,$
	$index$	$\mapsto i,$
	$values$	$\mapsto transaction[i].values[t],$
	$dependencyIndex$	$\mapsto 0,$
	$rollbackIndex$	$\mapsto 0,$
	$rollbackValues$	$\mapsto \langle \rangle,$
	$status$	$\mapsto Initializing]$

ELSE

$proposal[t]$

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

$\vee \wedge transaction[i].type = Rollback$

$\wedge \vee \wedge transaction[i].rollback \in \text{DOMAIN } transaction$

$\wedge transaction[transaction[i].rollback].type = Change$

$\wedge transaction' = [transaction \text{ EXCEPT } ![i].targets =$
 $\text{DOMAIN } transaction[transaction[i].rollback].values]$

$\wedge proposal' = [t \in \text{DOMAIN } proposal \mapsto$

IF $t \in \text{DOMAIN } transaction[transaction[i].rollback].values$ THEN

$proposal[t] @@ (i :=$	$[type$	$\mapsto Rollback,$
	$index$	$\mapsto i,$
	$rollback$	$\mapsto transaction[i].rollback,$

	$dependencyIndex \mapsto 0,$ $rollbackIndex \mapsto 0,$ $rollbackValues \mapsto \langle \rangle,$ $status \mapsto Initializing]$
--	---

ELSE

$proposal[t]$

$\vee \wedge \vee \wedge transaction[i].rollback \in \text{DOMAIN } transaction$

$\wedge transaction[transaction[i].rollback].type = Rollback$

$\vee transaction[i].rollback \notin \text{DOMAIN } transaction$

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

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

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

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

$\wedge \vee \wedge \forall t \in transaction[i].targets : proposal[t][i].status = Initialized$

$\wedge transaction' = [transaction \text{ EXCEPT } ![i].status = Initialized]$

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

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

$\vee \wedge \exists t \in transaction[i].targets : proposal[t][i].status = Failed$

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

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

Once the transaction has been *Initialized*, proceed to the Validate phase.

If any of the transaction's proposals depend on a *Serializable* transaction, verify the dependency has been *Validated* to preserve serializability before moving the transaction to the Validate phase.

$\vee \wedge transaction[i].status = Initialized$

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

$proposal[t][i].dependencyIndex \neq 0 \Rightarrow$

$(transaction[proposal[t][i].dependencyIndex].isolation = Serializable \Rightarrow$

$Phase(transaction[proposal[t][i].dependencyIndex].status) \geq Phase(Validated))$

$\wedge transaction' = [transaction \text{ EXCEPT } ![i].status = Validating]$

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

$\vee \wedge transaction[i].status = Validating$

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

$\wedge \vee \wedge \exists t \in transaction[i].targets : Phase(proposal[t][i].status) < Phase(Validating)$

$\wedge proposal' = [t \in \text{DOMAIN } proposal \mapsto$

IF $t \in transaction[i].targets$ THEN

$[proposal[t] \text{ EXCEPT } ![i].status = Validating]$

ELSE

$proposal[t]$

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

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

$\vee \wedge \forall t \in transaction[i].targets : proposal[t][i].status = Validated$

$\wedge transaction' = [transaction \text{ EXCEPT } ![i].status = Validated]$

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

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

$$\begin{aligned} & \vee \wedge \exists t \in \text{transaction}[i].\text{targets} : \text{proposal}[t][i].\text{status} = \text{Failed} \\ & \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{status} = \text{Failed}] \\ & \wedge \text{UNCHANGED } \langle \text{proposal} \rangle \end{aligned}$$

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.

$$\begin{aligned} & \vee \wedge \text{transaction}[i].\text{status} = \text{Validated} \\ & \wedge \forall t \in \text{transaction}[i].\text{targets} : \\ & \quad \text{proposal}[t][i].\text{dependencyIndex} \neq 0 \Rightarrow \\ & \quad \quad (\text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{isolation} = \text{Serializable} \Rightarrow \\ & \quad \quad \quad \text{Phase}(\text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{status}) \geq \text{Phase}(\text{Committed})) \\ & \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{status} = \text{Committing}] \\ & \wedge \text{UNCHANGED } \langle \text{proposal} \rangle \\ & \vee \wedge \text{transaction}[i].\text{status} = \text{Committing} \\ & \quad \text{Move the transaction's proposals to the } \text{Committing} \text{ state} \\ & \wedge \vee \wedge \exists t \in \text{transaction}[i].\text{targets} : \text{Phase}(\text{proposal}[t][i].\text{status}) < \text{Phase}(\text{Committing}) \\ & \quad \wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto \\ & \quad \quad \text{IF } t \in \text{transaction}[i].\text{targets} \text{ THEN} \\ & \quad \quad \quad [\text{proposal}[t] \text{ EXCEPT } ![i].\text{status} = \text{Committing}] \\ & \quad \quad \text{ELSE} \\ & \quad \quad \quad \text{proposal}[t]] \\ & \quad \wedge \text{UNCHANGED } \langle \text{transaction} \rangle \\ & \quad \text{If all proposals have been } \text{Committed}, \text{ mark the transaction } \text{Committed}. \\ & \vee \wedge \forall t \in \text{transaction}[i].\text{targets} : \text{proposal}[t][i].\text{status} = \text{Committed} \\ & \quad \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{status} = \text{Committed}] \\ & \quad \wedge \text{UNCHANGED } \langle \text{proposal} \rangle \end{aligned}$$

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.

$$\begin{aligned} & \vee \wedge \text{transaction}[i].\text{status} = \text{Committed} \\ & \wedge \forall t \in \text{transaction}[i].\text{targets} : \\ & \quad \text{proposal}[t][i].\text{dependencyIndex} \neq 0 \Rightarrow \\ & \quad \quad (\text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{isolation} = \text{Serializable} \Rightarrow \\ & \quad \quad \quad \text{Phase}(\text{transaction}[\text{proposal}[t][i].\text{dependencyIndex}].\text{status}) \geq \text{Phase}(\text{Applied})) \\ & \wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{status} = \text{Applying}] \\ & \wedge \text{UNCHANGED } \langle \text{proposal} \rangle \\ & \vee \wedge \text{transaction}[i].\text{status} = \text{Applying} \\ & \quad \text{Move the transaction's proposals to the } \text{Applying} \text{ state} \\ & \wedge \vee \wedge \exists t \in \text{transaction}[i].\text{targets} : \text{Phase}(\text{proposal}[t][i].\text{status}) < \text{Phase}(\text{Applying}) \\ & \quad \wedge \text{proposal}' = [t \in \text{DOMAIN } \text{proposal} \mapsto \\ & \quad \quad \text{IF } t \in \text{transaction}[i].\text{targets} \text{ THEN} \\ & \quad \quad \quad [\text{proposal}[t] \text{ EXCEPT } ![i].\text{status} = \text{Applying}] \\ & \quad \quad \text{ELSE} \\ & \quad \quad \quad \text{proposal}[t]] \end{aligned}$$

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.

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

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

$$![t].\text{configIndex} = \text{proposal}[t][i].\text{rollbackIndex},$$

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

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

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

While in the *Applying* state, apply the proposed changes to the target.

$$\vee \wedge \text{proposal}[t][i].\text{status} = \text{Applying}$$

$$\wedge \text{configuration}[t].\text{appliedIndex} = \text{proposal}[t][i].\text{dependencyIndex}$$

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

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

Model successful and failed target update requests.

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

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

If the proposal is a change, apply the change values to the target and update the configuration's applied index and values.

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

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

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

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

$$![t].\text{appliedValues} = \text{proposal}[t][i].\text{values} @@ \text{configuration}[t].\text{appliedValues}]$$

If the proposal is a rollback, apply the rollback values and update the configuration's applied values with the rolled back values.

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

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

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

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

$$![t].\text{appliedValues} = \text{proposal}[t][i].\text{rollbackValues} @@ \text{configuration}[t].\text{appliedValues}]$$

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

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

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

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

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

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

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

Init and next state predicates

$$\begin{aligned}
Init &\triangleq \\
&\wedge \textit{transaction} = \langle \rangle \\
&\wedge \textit{proposal} = [t \in \text{DOMAIN } Target \mapsto \\
&\quad [p \in \{\} \mapsto [\textit{status} \mapsto \textit{Initializing}]]] \\
&\wedge \textit{configuration} = [t \in \text{DOMAIN } Target \mapsto \\
&\quad [target \mapsto t, \\
&\quad \textit{status} \mapsto \textit{Pending}, \\
&\quad \textit{values} \mapsto \\
&\quad \quad [path \in \{\} \mapsto \\
&\quad \quad \quad [path \mapsto path, \\
&\quad \quad \quad \textit{value} \mapsto Nil, \\
&\quad \quad \quad \textit{index} \mapsto 0, \\
&\quad \quad \quad \textit{deleted} \mapsto FALSE]], \\
&\quad \textit{configIndex} \mapsto 0, \\
&\quad \textit{configTerm} \mapsto 0,
\end{aligned}$$

$$\begin{aligned}
& \text{proposedIndex} \mapsto 0, \\
& \text{committedIndex} \mapsto 0, \\
& \text{appliedIndex} \mapsto 0, \\
& \text{appliedTerm} \mapsto 0, \\
& \text{appliedValues} \mapsto \\
& \quad [\text{path} \in \{\} \mapsto \\
& \quad \quad [\text{path} \mapsto \text{path}, \\
& \quad \quad \text{value} \mapsto \text{Nil}, \\
& \quad \quad \text{index} \mapsto 0, \\
& \quad \quad \text{deleted} \mapsto \text{FALSE}]]]] \\
& \wedge \text{target} = [t \in \text{DOMAIN } \text{Target} \mapsto \\
& \quad [\text{path} \in \{\} \mapsto \\
& \quad \quad [\text{value} \mapsto \text{Nil}]]] \\
& \wedge \text{mastership} = [t \in \text{DOMAIN } \text{Target} \mapsto [\text{master} \mapsto \text{Nil}, \text{term} \mapsto 0]] \\
\text{Next} & \triangleq \\
& \vee \exists c \in \text{ValidChanges} : \\
& \quad \text{RequestChange}(c) \\
& \vee \exists t \in \text{DOMAIN } \text{transaction} : \\
& \quad \text{RequestRollback}(t) \\
& \vee \exists n \in \text{Node} : \\
& \quad \exists t \in \text{DOMAIN } \text{Target} : \\
& \quad \quad \text{SetMaster}(n, t) \\
& \quad \vee \exists t \in \text{DOMAIN } \text{Target} : \\
& \quad \quad \text{UnsetMaster}(t) \\
& \vee \exists n \in \text{Node} : \\
& \quad \exists t \in \text{DOMAIN } \text{transaction} : \\
& \quad \quad \text{ReconcileTransaction}(n, t) \\
& \vee \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) \\
& \vee \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{Isolation} & \triangleq \\
& \wedge \forall i, j \in \text{DOMAIN } \text{transaction} : \\
& \quad \vee j \leq i \\
& \quad \vee \text{transaction}[i].\text{targets} \cap \text{transaction}[j].\text{targets} = \{\} \\
& \quad \vee \text{transaction}[i].\text{isolation} \neq \text{Serializable} \\
& \quad \vee \text{Phase}(\text{transaction}[i].\text{status}) \geq \text{Phase}(\text{Committed}) \\
& \quad \vee \text{transaction}[j].\text{status} = \text{Failed} \\
& \quad \vee \text{Phase}(\text{transaction}[j].\text{status}) < \text{Phase}(\text{Committing})
\end{aligned}$$

$Order \triangleq \text{TRUE}$ *TODO* redefine order spec

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

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

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

\ * Modification History

\ * Last modified Sun Feb 06 23:16:24 PST 2022 by jordanhalterman

\ * Created Wed Sep 22 13:22:32 PDT 2021 by jordanhalterman