

---

MODULE *Config*

---

INSTANCE *Naturals*

INSTANCE *FiniteSets*

INSTANCE *Sequences*

INSTANCE *TLC*

---

An empty constant

CONSTANT *Nil*

Transaction type constants

CONSTANTS

*TransactionChange*,  
*TransactionRollback*

Transaction status constants

CONSTANTS

*TransactionPending*,  
*TransactionValidating*,  
*TransactionApplying*,  
*TransactionComplete*,  
*TransactionFailed*

Configuration status constants

CONSTANTS

*ConfigurationPending*,  
*ConfigurationInitializing*,  
*ConfigurationUpdating*,  
*ConfigurationComplete*,  
*ConfigurationFailed*

The set of all nodes

CONSTANT *Node*

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

Example:  $Target \triangleq$  [  
   $target1 \mapsto$  [  
     $path1 \mapsto \{ "value1", "value2" \}$ ,  
     $path2 \mapsto \{ "value2", "value3" \}$ ],  
   $target2 \mapsto$  [  
     $path2 \mapsto \{ "value3", "value4" \}$ ,  
     $path3 \mapsto \{ "value4", "value5" \}$ ]]

CONSTANT *Target*

```

ASSUME Nil ∈ STRING

ASSUME TransactionPending ∈ STRING
ASSUME TransactionValidating ∈ STRING
ASSUME TransactionApplying ∈ STRING
ASSUME TransactionComplete ∈ STRING
ASSUME TransactionFailed ∈ STRING

ASSUME ConfigurationPending ∈ STRING
ASSUME ConfigurationInitializing ∈ STRING
ASSUME ConfigurationUpdating ∈ STRING
ASSUME ConfigurationComplete ∈ STRING
ASSUME ConfigurationFailed ∈ STRING

ASSUME ∧ IsFiniteSet(Node)
      ∧ ∀ n ∈ Node :
        ∧ n ∉ DOMAIN Target
        ∧ n ∈ STRING

ASSUME ∧ ∀ t ∈ DOMAIN Target :
      ∧ t ∉ Node
      ∧ t ∈ STRING
      ∧ ∀ p ∈ DOMAIN Target[t] :
        IsFiniteSet(Target[t][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 TransactionType ::= type ∈
  { TransactionChange,
    TransactionRollback }

TYPE TransactionStatus ::= status ∈
  { TransactionPending,
    TransactionValidating,
    TransactionApplying,
    TransactionComplete,
    TransactionFailed }

TYPE Transaction  $\triangleq$  [
  type    ::= type ∈ TransactionType,
  index ::= index ∈ Nat,
  revision ::= revision ∈ Nat,
  atomic ::= atomic ∈ BOOLEAN ,
  sync    ::= sync ∈ BOOLEAN ,
  changes ::= [ target ∈ SUBSET (DOMAIN Target) ↦ [
    path ∈ SUBSET (DOMAIN Target[target]) ↦ [

```

```

    value ::= value ∈ STRING,
    delete ::= delete ∈ BOOLEAN ]]],
rollback ::= index ∈ Nat,
sources ::=
    target ∈ SUBSET (DOMAIN Target) ↦ [ path ∈ SUBSET (DOMAIN Target[target]) ↦ [
        value ::= value ∈ STRING,
        index ::= index ∈ Nat ]]],
status ::= status ∈ TransactionStatus]

TYPE ConfigurationStatus ::= status ∈
{ ConfigurationPending,
  ConfigurationInitializing,
  ConfigurationUpdating,
  ConfigurationComplete,
  ConfigurationFailed }

TYPE Configuration  $\triangleq$  [
  id      ::= id ∈ STRING,
  revision ::= revision ∈ Nat,
  target  ::= target ∈ STRING,
  paths  ::= [ path ∈ SUBSET (DOMAIN Target[target]) ↦ [
      value ::= value ∈ STRING,
      index  ::= index ∈ Nat,
      deleted ::= delete ∈ BOOLEAN ]],
  txIndex ::= txIndex ∈ Nat,
  syncIndex ::= syncIndex ∈ Nat,
  term     ::= term ∈ Nat,
  status  ::= status ∈ ConfigurationStatus]

```

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 configurations

VARIABLE *configuration*

A record of target states

VARIABLE *target*

A record of target masters

VARIABLE *master*

A history variable tracking past configuration changes

VARIABLE *history*

$vars \triangleq \langle transaction, configuration, master, target, history \rangle$

---

This section models mastership for the configuration service.

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

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

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

---

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

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

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

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

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

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

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

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.

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

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

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

```

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

$Change(c) \triangleq$

```

 $\wedge$  transaction' = transaction @@ (NextIndex > [
  type       $\mapsto$  TransactionChange,
  index      $\mapsto$  NextIndex,
  atomic     $\mapsto$  FALSE,
  sync       $\mapsto$  FALSE,
  changes    $\mapsto$  c,
  sources    $\mapsto$   $\langle \rangle$ ,
  status     $\mapsto$  TransactionPending])

 $\wedge$  UNCHANGED  $\langle configuration, master, target, history \rangle$ 

```

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

$Rollback(t) \triangleq$

```

 $\wedge$  transaction[t].type = TransactionChange
 $\wedge$  transaction' = transaction @@ (NextIndex > [
  type       $\mapsto$  TransactionRollback,
  index      $\mapsto$  NextIndex,
  atomic     $\mapsto$  FALSE,
  sync       $\mapsto$  FALSE,
  rollback   $\mapsto$  t,
  status     $\mapsto$  TransactionPending])

 $\wedge$  UNCHANGED  $\langle configuration, master, target, history \rangle$ 

```

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

Both types of transaction are reconciled in stages:

- \* Pending - waiting for prior transactions to complete
- \* Validating - validating the requested changes
- \* Applying - applying the changes to target configurations
- \* Complete - completed applying changes successfully
- \* Failed - failed applying changes

Reconcile a change transaction  
 $ReconcileChange(n, i) \triangleq$

If the transaction is Pending, begin validation if the prior transaction has already been applied. This simplifies concurrency control in the controller and guarantees transactions are applied to the configurations in sequential order.

$\vee \wedge transaction[i].status = TransactionPending$   
 $\wedge \vee \wedge i - 1 \in \text{DOMAIN } transaction$   
 $\wedge transaction[i - 1].status \in \{TransactionComplete, TransactionFailed\}$   
 $\vee i - 1 \notin \text{DOMAIN } transaction$   
 $\wedge transaction' = [transaction \text{ EXCEPT } ![i].status = TransactionValidating]$   
 $\wedge \text{UNCHANGED } \langle configuration, history \rangle$

If the transaction is in the Validating state, compute and validate the Configuration for each target.

$\vee \wedge transaction[i].status = TransactionValidating$   
 If validation fails any target, mark the transaction Failed.  
 If validation is successful, proceed to Applying.  
 $\wedge \exists valid \in \text{BOOLEAN} :$   
 IF  $valid$  THEN  
 $\wedge transaction' = [transaction \text{ EXCEPT } ![i].status = TransactionApplying]$   
 ELSE  
 $\wedge transaction' = [transaction \text{ EXCEPT } ![i].status = TransactionFailed]$   
 $\wedge \text{UNCHANGED } \langle configuration, history \rangle$

If the transaction is in the Applying state, update the Configuration for each target and Complete the transaction.

$\vee \wedge transaction[i].status = TransactionApplying$   
 Update the target configurations, adding the transaction index to each updated path  
 $\wedge configuration' = [$   
 $t \in \text{DOMAIN } Target \mapsto$   
 IF  $t \in \text{DOMAIN } transaction[i].changes$  THEN  
 $[configuration[t] \text{ EXCEPT }$   
 $!.paths = [p \in \text{DOMAIN } transaction[i].changes[t] \mapsto$   
 $[index \mapsto transaction[i].index,$   
 $value \mapsto transaction[i].changes[t][p].value,$   
 $deleted \mapsto transaction[i].changes[t][p].delete]]$   
 $@@ configuration[t].paths,$   
 $!.txIndex = transaction[i].index,$   
 $!.status = ConfigurationPending]$   
 ELSE  
 $configuration[t]]$   
 $\wedge history' = [r \in \text{DOMAIN } Target \mapsto Append(history[r], configuration'[r])]$

The transaction state is updated to include the source configuration modified.  
 The source configuration is used to optimize rollbacks.  
 Note that in a real-world implementation, the order of updates to the configuration and to add the source info to the transaction could have serious ramifications.  
 If one is updated without the other, rollbacks may not be possible.

$$\begin{aligned}
& \wedge transaction' = [transaction \text{ EXCEPT} \\
& \quad ![i].status = TransactionComplete, \\
& \quad ![i].sources = [t \in \text{DOMAIN } transaction[i].changes \mapsto \\
& \quad \quad \text{LET } updatePaths \triangleq \{p \in \text{DOMAIN } transaction[i].changes[t] : \\
& \quad \quad \quad \neg transaction[i].changes[t][p].delete\} \\
& \quad \text{IN } [p \in updatePaths \cap \text{DOMAIN } configuration[t].paths \mapsto configuration[t].paths[p]]]]
\end{aligned}$$

Reconcile a rollback transaction

$ReconcileRollback(n, i) \triangleq$

If the transaction is Pending, begin validation if the prior transaction has already been applied. This simplifies concurrency control in the controller and guarantees transactions are applied to the configurations in sequential order.

$$\begin{aligned}
& \vee \wedge transaction[i].status = TransactionPending \\
& \quad \wedge \vee \wedge i - 1 \in \text{DOMAIN } transaction \\
& \quad \quad \wedge transaction[i - 1].status \in \{TransactionComplete, TransactionFailed\} \\
& \quad \quad \vee i - 1 \notin \text{DOMAIN } transaction \\
& \quad \wedge transaction' = [transaction \text{ EXCEPT } ![i].status = TransactionValidating] \\
& \quad \wedge \text{UNCHANGED } \langle configuration, history \rangle
\end{aligned}$$

If the transaction is in the Validating state, validate the rollback.

A transaction can only be rolled back if:

1. The source transaction is in the log
2. The source transaction was applied successfully (did not fail validation)
3. The source transaction is the most recent change for each path is modified

$$\begin{aligned}
& \vee \wedge transaction[i].status = TransactionValidating \\
& \quad \wedge \vee \wedge transaction[transaction[i].rollback].status = TransactionComplete \\
& \quad \wedge \vee \wedge transaction[i].rollback \in \text{DOMAIN } transaction \\
& \quad \quad \text{Determine whether the source transaction is the most recent change} \\
& \quad \quad \text{by comparing the configuration path indexes to the transaction index.} \\
& \quad \wedge \text{LET } canRollback \triangleq \forall t \in \text{DOMAIN } transaction[transaction[i].rollback].changes : \\
& \quad \quad \quad \forall p \in \text{DOMAIN } transaction[transaction[i].rollback].changes[t] : \\
& \quad \quad \quad \quad configuration[t].paths[p].index = transaction[i].rollback
\end{aligned}$$

IN

IF  $canRollback$  THEN

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

ELSE

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

If the source transaction is not in the log, fail the rollback.

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

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

If the source transaction failed, fail the rollback.

$$\vee \wedge transaction[transaction[i].rollback].status = TransactionFailed$$

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

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

If the transaction is in the Applying state, roll back the Configuration for each target and Complete the transaction.

$\vee \wedge \text{transaction}[i].\text{status} = \text{TransactionApplying}$   
 Target configurations are rolled back by reverting to the source paths/values stored in the transaction when it was applied.  
 $\wedge \text{configuration}' = [$   
 $\quad t \in \text{DOMAIN Target} \mapsto$   
 $\quad \text{IF } t \in \text{DOMAIN transaction}[\text{transaction}[i].\text{rollback}].\text{changes} \text{ THEN}$   
 $\quad \quad \text{LET adds} \triangleq \{p \in \text{DOMAIN transaction}[\text{transaction}[i].\text{rollback}].\text{changes}[t] :$   
 $\quad \quad \quad \wedge p \notin \text{DOMAIN transaction}[\text{transaction}[i].\text{rollback}].\text{sources}[t]$   
 $\quad \quad \quad \wedge \neg \text{transaction}[\text{transaction}[i].\text{rollback}].\text{changes}[t][p].\text{delete}\}$   
 $\quad \quad \text{updates} \triangleq \{p \in \text{DOMAIN transaction}[\text{transaction}[i].\text{rollback}].\text{changes}[t] :$   
 $\quad \quad \quad \wedge p \in \text{DOMAIN transaction}[\text{transaction}[i].\text{rollback}].\text{sources}[t]$   
 $\quad \quad \quad \wedge \neg \text{transaction}[\text{transaction}[i].\text{rollback}].\text{changes}[t][p].\text{delete}\}$   
 $\quad \quad \text{removes} \triangleq \{p \in \text{DOMAIN transaction}[\text{transaction}[i].\text{rollback}].\text{changes}[t] :$   
 $\quad \quad \quad \wedge p \in \text{DOMAIN transaction}[\text{transaction}[i].\text{rollback}].\text{sources}[t]$   
 $\quad \quad \quad \wedge \text{transaction}[\text{transaction}[i].\text{rollback}].\text{changes}[t][p].\text{delete}\}$   
 $\quad \quad \text{changes} \triangleq \text{adds} \cup \text{updates} \cup \text{removes}$   
 $\quad \quad \text{unchanges} \triangleq \text{DOMAIN configuration}[t].\text{paths} \setminus \text{changes}$   
 $\quad \text{IN}$   
 $\quad \quad [\text{configuration}[t] \text{ EXCEPT}$   
 $\quad \quad \quad !.\text{paths} = [p \in \text{unchanges} \mapsto \text{configuration}[t].\text{paths}[p]]$   
 $\quad \quad \quad @@ [p \in \text{updates} \cup \text{removes} \mapsto$   
 $\quad \quad \quad \quad \text{transaction}[\text{transaction}[i].\text{rollback}].\text{sources}[t][p]],$   
 $\quad \quad \quad !.\text{txIndex} = \text{transaction}[i].\text{index},$   
 $\quad \quad \quad !.\text{status} = \text{ConfigurationPending}]$   
 $\quad \text{ELSE}$   
 $\quad \quad \text{configuration}[t]]$   
 $\wedge \text{history}' = [r \in \text{DOMAIN Target} \mapsto \text{Append}(\text{history}[r], \text{configuration}'[r])]$   
 $\wedge \text{transaction}' = [\text{transaction} \text{ EXCEPT } ![i].\text{status} = \text{TransactionComplete}]$

Reconcile a transaction in the transaction log

Transactions can be of one of two types: *Change* and *Rollback*.

The logic for processing different types of transactions differs.

$\text{ReconcileTransaction}(n, i) \triangleq$   
 $\quad \wedge \vee \wedge \text{transaction}[i].\text{type} = \text{TransactionChange}$   
 $\quad \quad \wedge \text{ReconcileChange}(n, i)$   
 $\quad \vee \wedge \text{transaction}[i].\text{type} = \text{TransactionRollback}$   
 $\quad \quad \wedge \text{ReconcileRollback}(n, i)$   
 $\quad \wedge \text{UNCHANGED } \langle \text{master}, \text{target} \rangle$

---

This section models the Configuration reconciler.

$\text{ReconcileConfiguration}(n, c) \triangleq$   
 $\quad \wedge \vee \wedge \text{configuration}[c].\text{status} = \text{ConfigurationPending}$   
 $\quad \quad \wedge \text{master}[\text{configuration}[c].\text{target}].\text{master} \neq \text{Nil}$   
 $\quad \quad \text{If the configuration is marked } \text{ConfigurationPending} \text{ and mastership}$



has changed (indicated by an increased mastership term), mark the configuration *ConfigurationInitializing* to force full re-synchronization.

$$\wedge \vee \wedge \text{master}[\text{configuration}[c].\text{target}].\text{term} > \text{configuration}[c].\text{term}$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![c].\text{status} = \text{ConfigurationInitializing},$$

$$![c].\text{term} = \text{master}[\text{configuration}[c].\text{target}].\text{term}]$$

$$\wedge \text{history}' = [\text{history} \text{ EXCEPT } ![c] = \text{Append}(\text{history}[c], \text{configuration}'[c])]$$

If the configuration is marked *ConfigurationPending* and the values have changed (determined by comparing the transaction index to the last *sync* index), mark the configuration *ConfigurationUpdating* to push the changes to the target.

$$\vee \wedge \text{master}[\text{configuration}[c].\text{target}].\text{term} = \text{configuration}[c].\text{term}$$

$$\wedge \text{configuration}[c].\text{syncIndex} < \text{configuration}[c].\text{txIndex}$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![c].\text{status} = \text{ConfigurationUpdating}]$$

$$\wedge \text{history}' = [\text{history} \text{ EXCEPT } ![c] = \text{Append}(\text{history}[c], \text{configuration}'[c])]$$

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

$$\vee \wedge \text{configuration}[c].\text{status} = \text{ConfigurationInitializing}$$

$$\wedge \text{master}[\text{configuration}[c].\text{target}].\text{master} = n$$

Merge the configuration paths with the target paths, removing paths that have been marked deleted

$$\wedge \text{LET } \text{deletePaths} \triangleq \{p \in \text{DOMAIN } \text{configuration}[c].\text{paths} : \text{configuration}[c].\text{paths}[p].\text{deleted}\}$$

$$\text{configPaths} \triangleq \text{DOMAIN } \text{configuration}[c].\text{paths} \setminus \text{deletePaths}$$

$$\text{targetPaths} \triangleq \text{DOMAIN } \text{target}[\text{configuration}[c].\text{target}] \setminus \text{deletePaths}$$

IN

$$\wedge \text{target}' = [\text{target} \text{ EXCEPT } ![c].\text{target} =$$

$$[p \in \text{configPaths} \mapsto [\text{value} \mapsto \text{configuration}[c].\text{paths}[p]]]$$

$$\text{@@ } [p \in \text{targetPaths} \mapsto \text{target}[\text{configuration}[c].\text{target}][p]]]$$

Set the configuration's status to Complete

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } ![c].\text{status} = \text{ConfigurationComplete},$$

$$![c].\text{syncIndex} = \text{configuration}[c].\text{txIndex}]$$

$$\wedge \text{history}' = [\text{history} \text{ EXCEPT } ![c] = \text{Append}(\text{history}[c], \text{configuration}'[c])]$$

If the configuration is marked *ConfigurationUpdating*, we only need to push paths that have changed since the target was initialized or last updated by the controller. The set of changes made since the last synchronization are identified by comparing the index of each path-value to the last synchronization index, *syncIndex*

$$\vee \wedge \text{configuration}[c].\text{status} = \text{ConfigurationUpdating}$$

$$\wedge \text{master}[\text{configuration}[c].\text{target}].\text{master} = n$$

Compute the set of updated and deleted paths by comparing their indexes to the target's last sync index.

$$\wedge \text{LET } \text{updatePaths} \triangleq \{p \in \text{DOMAIN } \text{configuration}[c].\text{paths} :$$

$$\text{configuration}[c].\text{paths}[p].\text{index} > \text{configuration}[c].\text{syncIndex}\}$$

$$\text{deletePaths} \triangleq \{p \in \text{updatePaths} : \text{configuration}[c].\text{paths}[p].\text{deleted}\}$$

$$\text{configPaths} \triangleq \text{updatePaths} \setminus \text{deletePaths}$$

$$\text{targetPaths} \triangleq \text{DOMAIN } \text{target}[\text{configuration}[c].\text{target}] \setminus \text{deletePaths}$$

IN

Update the target paths by adding/updating paths that have changed and removing paths that have been deleted since the last *sync*.

$$\begin{aligned} \wedge target' &= [target \text{ EXCEPT } ![configuration[c].target] = \\ &\quad [p \in configPaths \mapsto configuration[c].paths[p]] \\ &\quad @@ [p \in targetPaths \mapsto target[configuration[c].target][p]]] \\ \wedge configuration' &= [configuration \text{ EXCEPT } ![c].status = ConfigurationComplete, \\ &\quad ![c].syncIndex = configuration[c].txIndex] \\ \wedge history' &= [history \text{ EXCEPT } ![c] = Append(history[c], configuration'[c])] \end{aligned}$$

If the configuration is not already *ConfigurationPending* and mastership has been lost revert it. This can occur when the connection to the target has been lost and the mastership is no longer valid.

$$\begin{aligned} \vee \wedge configuration[c].status &\neq ConfigurationPending \\ \wedge master[configuration[c].target].master &= Nil \\ \wedge configuration' &= [configuration \text{ EXCEPT } ![c].status = ConfigurationPending] \\ \wedge history' &= [history \text{ EXCEPT } ![c] = Append(history[c], configuration'[c])] \\ \wedge UNCHANGED \langle target \rangle \\ \wedge UNCHANGED \langle transaction, master \rangle \end{aligned}$$


---

*Init* and next state predicates

*Init*  $\triangleq$

$$\begin{aligned} \wedge transaction &= \langle \rangle \\ \wedge configuration &= [t \in \text{DOMAIN } Target \mapsto \\ &\quad [target \mapsto t, \\ &\quad paths \mapsto \\ &\quad \quad [path \in \{\} \mapsto \\ &\quad \quad \quad [path \mapsto path, \\ &\quad \quad \quad value \mapsto Nil, \\ &\quad \quad \quad index \mapsto 0, \\ &\quad \quad \quad deleted \mapsto FALSE]], \\ &\quad txIndex \mapsto 0, \\ &\quad syncIndex \mapsto 0, \\ &\quad term \mapsto 0, \\ &\quad status \mapsto ConfigurationPending]] \\ \wedge target &= [t \in \text{DOMAIN } Target \mapsto \\ &\quad [path \in \{\} \mapsto \\ &\quad \quad [value \mapsto Nil]]] \\ \wedge master &= [t \in \text{DOMAIN } Target \mapsto [master \mapsto Nil, term \mapsto 0]] \\ \wedge history &= [t \in \text{DOMAIN } Target \mapsto \langle \rangle] \end{aligned}$$

*Next*  $\triangleq$

$$\begin{aligned} \vee \exists c \in ValidChanges : \\ \quad Change(c) \\ \vee \exists t \in \text{DOMAIN } transaction : \\ \quad Rollback(t) \end{aligned}$$

$$\begin{aligned}
& \forall \exists n \in \text{Node} : \\
& \quad \exists t \in \text{DOMAIN } \text{Target} : \\
& \quad \quad \text{SetMaster}(n, t) \\
& \forall \exists t \in \text{DOMAIN } \text{Target} : \\
& \quad \text{UnsetMaster}(t) \\
& \forall \exists n \in \text{Node} : \\
& \quad \exists t \in \text{DOMAIN } \text{transaction} : \\
& \quad \quad \text{ReconcileTransaction}(n, t) \\
& \forall \exists n \in \text{Node} : \\
& \quad \exists c \in \text{DOMAIN } \text{configuration} : \\
& \quad \quad \text{ReconcileConfiguration}(n, c) \\
\text{Spec} & \triangleq \text{Init} \wedge \Box[\text{Next}]_{\text{vars}} \\
\text{Order} & \triangleq \\
& \wedge \forall a, b \in \text{DOMAIN } \text{transaction} : \\
& \quad \text{transaction}[a].\text{index} > \text{transaction}[b].\text{index} \Rightarrow \\
& \quad (\text{transaction}[a].\text{status} \in \{\text{TransactionComplete}, \text{TransactionFailed}\} \Rightarrow \\
& \quad \quad \text{transaction}[b].\text{status} \in \{\text{TransactionComplete}, \text{TransactionFailed}\}) \\
& \wedge \forall t \in \text{DOMAIN } \text{Target} : \\
& \quad \forall c \in \text{DOMAIN } \text{history}[t] : \\
& \quad \quad \wedge \text{configuration}[t].\text{txIndex} \geq \text{history}[t][c].\text{txIndex} \\
& \quad \quad \wedge \text{configuration}[t].\text{syncIndex} \geq \text{history}[t][c].\text{syncIndex} \\
\text{THEOREM } \text{Safety} & \triangleq \text{Spec} \Rightarrow \Box \text{Order} \\
\text{Completion} & \triangleq \forall i \in \text{DOMAIN } \text{transaction} : \\
& \quad \text{transaction}[i].\text{status} \in \{\text{TransactionComplete}, \text{TransactionFailed}\} \\
\text{THEOREM } \text{Liveness} & \triangleq \text{Spec} \Rightarrow \Diamond \text{Completion}
\end{aligned}$$


---

\ \* Modification History  
\ \* Last modified Wed Jan 19 00:12:18 PST 2022 by jordanhalterman  
\ \* Created Wed Sep 22 13:22:32 PDT 2021 by jordanhalterman