
MODULE *Transaction*

This module specifies the logic for the transaction controller. This controller is responsible for doing the vast majority of the processing in *μONOS Config*.

The transaction controller processes transactions as they're appended to the transaction *log* by the northbound server. Each transaction may go through up to two phases: *Change* and *Rollback*. All transactions are initially processed through the *Change* phase after being appended to the *log*. Once they've been committed to the internal configuration, transactions can be rolled back by request via the northbound server. Additionally, there are two stages through which the transaction must progress within each phase. In the *Commit* stage, the controller will validate the change and commit it to the internal configuration, making it immediately queryable via the northbound *API*. Once a change has been committed, it will be sent to the target during the *Apply* stage.

The transaction controller provides a few important guarantees for processing transactions : –

- Changes and rollbacks will always be committed to the configuration before they're applied to the *target*
- Changes will always be committed and applied in the order in which they were appended to the transaction *log*
- *Rollbacks* will always be committed and applied in reverse chronological order
- Transactions will always be applied in the order in which they were *committed*
- If a transaction passes validation but is rejected by the target, the rejected transaction and any later changes that are pending must be rolled back before any subsequent change can be applied (due to the semantics of guarantee $\neq 2$)

INSTANCE *Naturals*

INSTANCE *FiniteSets*

INSTANCE *Sequences*

INSTANCE *TLC*

An empty constant

CONSTANT *Nil*

Transaction phase constants

CONSTANTS

Change,

Rollback

Transaction stage constants

CONSTANTS

Commit,

Apply

Status constants representing the state associated with a transaction state (commit/apply)

CONSTANTS

Pending,

InProgress,

Complete,
Aborted,
Canceled,
Failed

$Status \triangleq \{Pending, InProgress, Complete, Aborted, Canceled, Failed\}$

$Done \triangleq \{Complete, Aborted, Canceled, Failed\}$

The set of all nodes

CONSTANT *Node*

The set of possible paths and values

CONSTANT *Path, Value*

$Empty \triangleq [p \in \{\} \mapsto Nil]$

The state variables that are managed by other controllers.

VARIABLES

configuration,
mastership,
conns,
target

A transaction log. Changes are appended to the log and can be rolled back by modifying the state of a transaction.

VARIABLE *transactions*

A history of transaction change/rollback commit/apply events used for model checking.

VARIABLE *history*

Transactions are managed externally by two operations: append and rollback. Transactions are appended in the *Change* phase. Once a transaction has been committed successfully, it can be rolled back by simply moving it to the Rollback phase.

Add a change for revision 'i' to the transaction log

$AppendChange(i) \triangleq$
 $\wedge Len(transactions) = i - 1$
 $\wedge \exists p \in Path, v \in Value :$
 LET *transaction* \triangleq [
 index $\mapsto Len(transactions) + 1,$
 phase $\mapsto Change,$
 change \mapsto [
 index $\mapsto Len(transactions) + 1,$
 ordinal $\mapsto 0,$
 values $\mapsto (p := v),$

Again, we model partial commits due to failures during the commit process.

$$\begin{aligned}
& \wedge \vee \wedge \text{LET } \text{ordinal} \triangleq \text{configuration.committed.ordinal} + 1 \\
& \quad \text{values} \triangleq \text{transactions}[i].\text{change.values} @@ \text{configuration.committed.values} \\
& \text{IN} \\
& \quad \wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } \begin{array}{l} \text{!.committed.index} = i, \\ \text{!.committed.change} = i, \\ \text{!.committed.revision} = i, \\ \text{!.committed.ordinal} = \text{ordinal}, \\ \text{!.committed.values} = \text{values} \end{array} \\
& \quad \wedge \text{history}' = \text{Append}(\text{history}, [\\
& \quad \quad \text{phase} \mapsto \text{Change}, \\
& \quad \quad \text{event} \mapsto \text{Commit}, \\
& \quad \quad \text{index} \mapsto i, \\
& \quad \quad \text{status} \mapsto \text{Complete}]) \\
& \quad \text{To ensure the change will be applied in the order in which it was} \\
& \quad \text{committed, store the sequential ordinal to be used to sequence changes.} \\
& \quad \wedge \vee \text{transactions}' = [\text{transactions} \text{ EXCEPT } \begin{array}{l} \text{![i].change.commit} = \text{Complete}, \\ \text{![i].change.ordinal} = \text{ordinal} \end{array} \\
& \quad \vee \text{UNCHANGED } \langle \text{transactions} \rangle \\
& \quad \text{In the event of a validation failure, the transaction status is updated} \\
& \quad \text{*before* the configuration indexes. This is safe because the configuration} \\
& \quad \text{values are not updated with invalid change values. Updating the transaction} \\
& \quad \text{status first ensures important error information is retained in the event} \\
& \quad \text{of a failure while processing the change.} \\
& \quad \vee \wedge \text{transactions}' = [\text{transactions} \text{ EXCEPT } \begin{array}{l} \text{![i].change.commit} = \text{Failed}, \\ \text{![i].change.apply} = \text{Canceled} \end{array} \\
& \quad \wedge \text{history}' = \text{Append}(\text{history}, [\\
& \quad \quad \text{phase} \mapsto \text{Change}, \\
& \quad \quad \text{event} \mapsto \text{Commit}, \\
& \quad \quad \text{index} \mapsto i, \\
& \quad \quad \text{status} \mapsto \text{Failed}]) \\
& \quad \wedge \vee \text{configuration}' = [\text{configuration} \text{ EXCEPT } \begin{array}{l} \text{!.committed.index} = i, \\ \text{!.committed.change} = i \end{array} \\
& \quad \vee \text{UNCHANGED } \langle \text{configuration} \rangle \\
& \quad \text{If the configuration was updated but the commit is still } \text{InProgress}, \text{ move} \\
& \quad \text{it to } \text{Complete}. \\
& \quad \vee \wedge \text{configuration.committed.change} = i \\
& \quad \quad \wedge \text{transactions}' = [\text{transactions} \text{ EXCEPT } \begin{array}{l} \text{![i].change.commit} = \text{Complete}, \\ \text{![i].change.ordinal} = \text{configuration.committed.ordinal} \end{array} \\
& \quad \quad \wedge \text{UNCHANGED } \langle \text{configuration}, \text{history} \rangle \\
& \quad \text{If the change commit was marked } \text{Failed} \text{ but a failure occurred before the} \\
& \quad \text{configuration could be updated, update the configuration indexes to unblock} \\
& \quad \text{subsequent transactions.} \\
& \quad \vee \wedge \text{transactions}[i].\text{change.commit} = \text{Failed} \\
& \quad \quad \wedge \text{configuration.committed.change} < i
\end{aligned}$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } !.\text{committed.index} = i, \\ !.\text{committed.change} = i] \\ \wedge \text{UNCHANGED } \langle \text{transactions}, \text{history} \rangle$$

Apply transaction 'i' change on node 'n'

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

$$\vee \wedge \text{transactions}[i].\text{change.apply} = \text{Pending}$$

To ensure changes are applied in the same order in which they were committed, we use the change ordinal to serialize application of changes/rollbacks to the target.

$$\wedge \vee \wedge \text{configuration.applied.ordinal} = \text{transactions}[i].\text{change.ordinal} - 1 \\ \wedge \text{configuration.applied.target} \neq i \\ \wedge \text{configuration.applied.index} \in \text{DOMAIN } \text{transactions} \Rightarrow \\ \vee \wedge \text{configuration.applied.target} = \text{configuration.applied.index} \\ \wedge \text{transactions}[\text{configuration.applied.index}].\text{change.apply} \in \text{Done} \\ \vee \wedge \text{configuration.applied.target} < \text{configuration.applied.index} \\ \wedge \text{transactions}[\text{configuration.applied.index}].\text{rollback.apply} \in \text{Done}$$

Just checking if the previous change/rollback has been committed is not enough to guarantee sequential ordering. In the event of a target failure, gaps in the log need to be accounted for. We use the applied revision to determine whether the prior change/rollback (the rollback index) was successful.

$$\wedge \vee \wedge \text{configuration.applied.revision} = \text{transactions}[i].\text{rollback.index} \\ \wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } !.\text{applied.target} = i] \\ \wedge \text{history}' = \text{Append}(\text{history}, [\\ \text{phase} \mapsto \text{Change}, \\ \text{event} \mapsto \text{Apply}, \\ \text{index} \mapsto i, \\ \text{status} \mapsto \text{InProgress}]) \\ \wedge \vee \text{transactions}' = [\text{transactions} \text{ EXCEPT } ![i].\text{change.apply} = \text{InProgress}] \\ \vee \text{UNCHANGED } \langle \text{transactions} \rangle$$

In the event that a prior change apply *Failed*, all subsequent changes must be *Aborted* and ultimately rolled back. This forces administrator intervention in the unlikely event of a mismatch between the config and target configuration models, in which case the change may pass validation but fail being applied to the target.

$$\vee \wedge \text{configuration.applied.revision} < \text{transactions}[i].\text{rollback.index} \\ \wedge \text{transactions}' = [\text{transactions} \text{ EXCEPT } ![i].\text{change.apply} = \text{Aborted}] \\ \wedge \text{history}' = \text{Append}(\text{history}, [\\ \text{phase} \mapsto \text{Change}, \\ \text{event} \mapsto \text{Apply}, \\ \text{index} \mapsto i, \\ \text{status} \mapsto \text{Aborted}]) \\ \wedge \vee \text{LET } \text{ordinal} \triangleq \text{transactions}[i].\text{change.ordinal} \\ \text{IN } \text{configuration}' = [\text{configuration} \text{ EXCEPT } !.\text{applied.target} = i,$$

$!.applied.index = i,$
 $!.applied.ordinal = ordinal]$

$\vee \text{ UNCHANGED } \langle configuration \rangle$

If the configuration was updated but the apply is still *Pending*, move it to *InProgress*.

$\vee \wedge configuration.applied.target = i$
 $\wedge transactions' = [transactions \text{ EXCEPT } ![i].change.apply = InProgress]$
 $\wedge \text{ UNCHANGED } \langle configuration, history \rangle$
 $\wedge \text{ UNCHANGED } \langle target \rangle$

$\vee \wedge transactions[i].change.apply = InProgress$

If the change has not yet been applied, attempt to apply it to the target.

This logic models the possibility of failures when applying changes to the *target* – despite them having already been validated - providing administrators an avenue to safely reconcile errors caused by mismatches between the system configuration models or between a model and the real-world implementation of it.

$\wedge \vee \wedge configuration.applied.ordinal \neq transactions[i].change.ordinal$

In the event of a node or target restart or another failure resulting in a *mastership* change, the configuration must be re-pushed to the target before the transaction controller can resume processing transactions to be applied to the target.

If the configuration is still being synchronized with the target, wait for the synchronization to be complete. The configuration state must be *Complete* in the current master's term before the change can be applied.

$\wedge configuration.state = Complete$
 $\wedge configuration.term = mastership.term$
 $\wedge conns[n].id = mastership.conn$
 $\wedge conns[n].connected$
 $\wedge target.running$

The change is successfully applied to the target.

$\wedge \vee \wedge \text{ LET } ordinal \triangleq transactions[i].change.ordinal$
 $values \triangleq transactions[i].change.values @@ configuration.applied.values$

IN

$\wedge target' = [target \text{ EXCEPT } !.values = transactions[i].change.values @@ target.values]$
 $\wedge configuration' = [configuration \text{ EXCEPT } !.applied.index = i,$
 $!.applied.ordinal = ordinal,$
 $!.applied.revision = i,$
 $!.applied.values = values]$

$\wedge history' = \text{Append}(history, [$
 $phase \mapsto Change,$
 $event \mapsto Apply,$
 $index \mapsto i,$
 $status \mapsto Complete])$

$\wedge \vee transactions' = [transactions \text{ EXCEPT } ![i].change.apply = Complete]$
 $\vee \text{ UNCHANGED } \langle transactions \rangle$

The target rejects the change with an unexpected error.

$$\begin{aligned}
& \text{event} \mapsto \text{Commit}, \\
& \text{index} \mapsto i, \\
& \text{status} \mapsto \text{InProgress}) \\
& \wedge \vee \text{transactions}' = [\text{transactions} \text{ EXCEPT } ![i].\text{rollback.commit} = \text{InProgress}] \\
& \quad \vee \text{UNCHANGED } \langle \text{transactions} \rangle \\
& \vee \wedge \text{configuration.committed.target} = \text{transactions}[i].\text{rollback.index} \\
& \quad \wedge \text{transactions}' = [\text{transactions} \text{ EXCEPT } ![i].\text{rollback.commit} = \text{InProgress}] \\
& \quad \wedge \text{UNCHANGED } \langle \text{configuration}, \text{history} \rangle \\
& \vee \wedge \text{transactions}[i].\text{rollback.commit} = \text{InProgress} \\
& \quad \text{We do not model validation here under the assumption the prior state} \\
& \quad \text{to which we are rolling back was already validated.} \\
& \wedge \vee \wedge \text{configuration.committed.revision} = i \\
& \quad \text{When completing the commit, the configuration status must be updated} \\
& \quad \text{before the rollback commit can be marked } \textit{Complete} \text{ to avoid multiple} \\
& \quad \text{transactions being processed concurrently.} \\
& \wedge \text{LET } \text{ordinal} \triangleq \text{configuration.committed.ordinal} + 1 \\
& \quad \text{revision} \triangleq \text{transactions}[i].\text{rollback.index} \\
& \quad \text{values} \triangleq \text{transactions}[i].\text{rollback.values} @@ \text{configuration.committed.values} \\
& \text{IN} \\
& \quad \wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } \begin{array}{ll} !.\text{committed.index} & = i, \\ !.\text{committed.ordinal} & = \text{ordinal}, \\ !.\text{committed.revision} & = \text{revision}, \\ !.\text{committed.values} & = \text{values} \end{array}] \\
& \quad \wedge \text{history}' = \text{Append}(\text{history}, [\\
& \quad \quad \text{phase} \mapsto \text{Rollback}, \\
& \quad \quad \text{event} \mapsto \text{Commit}, \\
& \quad \quad \text{index} \mapsto i, \\
& \quad \quad \text{status} \mapsto \text{Complete}]) \\
& \quad \text{Model partial commits due to failures during processing.} \\
& \quad \wedge \vee \text{transactions}' = [\text{transactions} \text{ EXCEPT } \begin{array}{ll} ![i].\text{rollback.commit} & = \text{Complete}, \\ ! & [i].\text{rollback.ordinal} = \text{ordinal} \end{array}] \\
& \quad \vee \text{UNCHANGED } \langle \text{transactions} \rangle \\
& \quad \text{In the event of a partial commit, } \textit{Complete} \text{ the rollback commit.} \\
& \vee \wedge \text{configuration.committed.revision} = \text{transactions}[i].\text{rollback.index} \\
& \quad \wedge \text{transactions}' = [\text{transactions} \text{ EXCEPT } \begin{array}{ll} ![i].\text{rollback.commit} & = \text{Complete}, \\ ! & [i].\text{rollback.ordinal} = \text{configuration.committed.ordinal} \end{array}] \\
& \quad \wedge \text{UNCHANGED } \langle \text{configuration}, \text{history} \rangle \\
& \quad \text{Apply transaction 'i' rollback on node 'n'} \\
& \text{ApplyRollback}(n, i) \triangleq \\
& \quad \vee \wedge \text{transactions}[i].\text{rollback.apply} = \text{Pending} \\
& \quad \text{Before the rollback can be applied, any changes that are } \textit{Pending} \text{ or } \textit{InProgress} \\
& \quad \text{must first be canceled.} \\
& \quad \text{If the change is } \textit{Pending} \text{ apply, mark the apply stage } \textit{Aborted}. \\
& \quad \wedge \vee \wedge \text{transactions}[i].\text{change.apply} = \text{Pending}
\end{aligned}$$

The change cannot be aborted until the previous scheduled change/rollback is complete.

$$\begin{aligned}
& \wedge \text{configuration.applied.ordinal} = \text{transactions}[i].\text{change.ordinal} - 1 \\
& \wedge \text{configuration.applied.target} \neq i \\
& \wedge \text{configuration.applied.index} \in \text{DOMAIN } \text{transactions} \Rightarrow \\
& \quad \vee \wedge \text{configuration.applied.target} = \text{configuration.applied.index} \\
& \quad \quad \wedge \text{transactions}[\text{configuration.applied.index}].\text{change.apply} \in \text{Done} \\
& \quad \vee \wedge \text{configuration.applied.target} < \text{configuration.applied.index} \\
& \quad \quad \wedge \text{transactions}[\text{configuration.applied.index}].\text{rollback.apply} \in \text{Done} \\
& \wedge \text{transactions}' = [\text{transactions} \text{ EXCEPT } ![i].\text{change.apply} = \text{Aborted}] \\
& \wedge \text{history}' = \text{Append}(\text{history}, [\\
& \quad \text{phase} \mapsto \text{Change}, \\
& \quad \text{event} \mapsto \text{Apply}, \\
& \quad \text{index} \mapsto i, \\
& \quad \text{status} \mapsto \text{Aborted}]) \\
& \wedge \vee \text{configuration}' = [\text{configuration} \text{ EXCEPT } !.\text{applied.target} = i, \\
& \quad !.\text{applied.index} = i, \\
& \quad !.\text{applied.ordinal} = \text{transactions}[i].\text{change.ordinal}] \\
& \vee \text{UNCHANGED } \langle \text{configuration} \rangle
\end{aligned}$$

If the change apply is *InProgress*, mark the apply stage *Failed* rather than aborting it. This is necessary to indicate that the change may or may not have been applied to the target. Since the apply was already in progress, a prior step may have already attempted to push the change to the target, and a failure during that attempt could have left the system in an inconsistent state. Once the change apply is *InProgress*, it must be rolled back completely through the apply phase even if it was never marked *Complete*.

$$\begin{aligned}
& \vee \wedge \text{transactions}[i].\text{change.apply} = \text{InProgress} \\
& \quad \wedge \text{configuration.applied.ordinal} \neq \text{transactions}[i].\text{change.ordinal} \\
& \quad \wedge \text{transactions}' = [\text{transactions} \text{ EXCEPT } ![i].\text{change.apply} = \text{Failed}] \\
& \quad \wedge \text{history}' = \text{Append}(\text{history}, [\\
& \quad \quad \text{phase} \mapsto \text{Change}, \\
& \quad \quad \text{event} \mapsto \text{Apply}, \\
& \quad \quad \text{index} \mapsto i, \\
& \quad \quad \text{status} \mapsto \text{Failed}]) \\
& \quad \wedge \vee \text{configuration}' = [\text{configuration} \text{ EXCEPT } !.\text{applied.index} = i, \\
& \quad \quad !.\text{applied.ordinal} = \text{transactions}[i].\text{change.ordinal}] \\
& \quad \vee \text{UNCHANGED } \langle \text{configuration} \rangle
\end{aligned}$$

If the transaction was *Aborted* or *Failed* but the configuration status hasn't been updated, update the configuration status to unblock subsequent changes/rollbacks. This can happen in the event a failure occurs while applying the change.

$$\begin{aligned}
& \vee \wedge \text{transactions}[i].\text{change.apply} \in \{\text{Aborted}, \text{Failed}\} \\
& \quad \wedge \text{configuration.applied.ordinal} < \text{transactions}[i].\text{change.ordinal} \\
& \quad \wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } !.\text{applied.target} = i, \\
& \quad \quad !.\text{applied.index} = i, \\
& \quad \quad !.\text{applied.ordinal} = \text{transactions}[i].\text{change.ordinal}] \\
& \quad \wedge \text{UNCHANGED } \langle \text{transactions}, \text{history} \rangle
\end{aligned}$$

Finally, the transaction's change phase really is done (based on the applied ordinal),
 apply the rollback in commit order (by rollback ordinal).

$$\vee \wedge \text{transactions}[i].\text{change.apply} \in \text{Done}$$

$$\wedge \text{configuration.applied.ordinal} = \text{transactions}[i].\text{rollback.ordinal} - 1$$

Model partial failures that can occur while transitioning the transaction.

$$\wedge \vee \wedge \text{configuration.applied.target} \neq \text{transactions}[i].\text{rollback.index}$$

$$\wedge \vee \wedge \text{configuration.applied.index} = i$$

$$\wedge \text{transactions}[\text{configuration.applied.index}].\text{change.apply} \in \text{Done}$$

$$\vee \wedge \text{configuration.applied.index} > i$$

$$\wedge \text{transactions}[\text{configuration.applied.index}].\text{rollback.apply} \in \text{Done}$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } !.\text{applied.target} = \text{transactions}[i].\text{rollback.index}]$$

$$\wedge \text{history}' = \text{Append}(\text{history}, [$$

$$\text{phase} \mapsto \text{Rollback},$$

$$\text{event} \mapsto \text{Apply},$$

$$\text{index} \mapsto i,$$

$$\text{status} \mapsto \text{InProgress}])$$

$$\wedge \vee \text{transactions}' = [\text{transactions} \text{ EXCEPT } ![i].\text{rollback.apply} = \text{InProgress}]$$

$$\vee \text{UNCHANGED } \langle \text{transactions} \rangle$$

A failure left the system in an inconsistent state, resume the transaction
 by moving it to *InProgress*.

$$\vee \wedge \text{configuration.applied.target} = \text{transactions}[i].\text{rollback.index}$$

$$\wedge \text{transactions}' = [\text{transactions} \text{ EXCEPT } ![i].\text{rollback.apply} = \text{InProgress}]$$

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

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

$$\vee \wedge \text{transactions}[i].\text{rollback.apply} = \text{InProgress}$$

If this transaction has not yet been applied, attempt to apply it.

$$\wedge \vee \wedge \text{configuration.applied.ordinal} \neq \text{transactions}[i].\text{rollback.ordinal}$$

In the event of a node or target restart or another failure resulting in a
mastership change, the configuration must be re-pushed to the target before
 the transaction controller can resume processing transactions to be applied
 to the target.

If the configuration is still being synchronized with the target, wait for
 the synchronization to be complete. The configuration state must be *Complete*
 in the current master's term before the change can be applied.

$$\wedge \text{configuration.state} = \text{Complete}$$

$$\wedge \text{configuration.term} = \text{mastership.term}$$

$$\wedge \text{conns}[n].\text{id} = \text{mastership.conn}$$

$$\wedge \text{conns}[n].\text{connected}$$

$$\wedge \text{target.running}$$

$$\wedge \text{LET } \text{ordinal} \triangleq \text{transactions}[i].\text{rollback.ordinal}$$

$$\text{revision} \triangleq \text{transactions}[i].\text{rollback.index}$$

$$\text{values} \triangleq \text{transactions}[i].\text{rollback.values} @@ \text{configuration.applied.values}$$

IN

$$\wedge \text{target}' = [\text{target} \text{ EXCEPT } !.\text{values} = \text{transactions}[i].\text{rollback.values} @@ \text{target.values}]$$

$$\wedge \text{configuration}' = [\text{configuration} \text{ EXCEPT } !.\text{applied.index} = i,$$

$$\begin{aligned}
& \text{!.applied.ordinal} = \text{ordinal}, \\
& \text{!.applied.revision} = \text{revision}, \\
& \text{!.applied.values} = \text{values}] \\
& \wedge \text{history}' = \text{Append}(\text{history}, [\\
& \quad \text{phase} \mapsto \text{Rollback}, \\
& \quad \text{event} \mapsto \text{Apply}, \\
& \quad \text{index} \mapsto i, \\
& \quad \text{status} \mapsto \text{Complete}]) \\
& \wedge \vee \text{transactions}' = [\text{transactions EXCEPT } ![i].\text{rollback.apply} = \text{Complete}] \\
& \quad \vee \text{UNCHANGED } \langle \text{transactions} \rangle \\
& \text{If the change has been applied, update the transaction status.} \\
& \vee \wedge \text{configuration.applied.ordinal} = \text{transactions}[i].\text{rollback.ordinal} \\
& \wedge \text{configuration.applied.revision} = \text{transactions}[i].\text{rollback.index} \\
& \wedge \text{transactions}' = [\text{transactions EXCEPT } ![i].\text{rollback.apply} = \text{Complete}] \\
& \wedge \text{UNCHANGED } \langle \text{configuration}, \text{target}, \text{history} \rangle
\end{aligned}$$

$$\begin{aligned}
& \text{Reconcile transaction 'i' rollback on node 'n'} \\
& \text{ReconcileRollback}(n, i) \triangleq \\
& \quad \wedge \text{transactions}[i].\text{phase} = \text{Rollback} \\
& \quad \wedge \vee \wedge \text{CommitRollback}(n, i) \\
& \quad \quad \wedge \text{UNCHANGED } \langle \text{target} \rangle \\
& \quad \vee \wedge \text{transactions}[i].\text{rollback.commit} = \text{Complete} \\
& \quad \wedge \text{ApplyRollback}(n, i)
\end{aligned}$$

$$\begin{aligned}
& \text{Reconcile transaction 'i' on node 'n'} \\
& \text{ReconcileTransaction}(n, i) \triangleq \\
& \quad \wedge i \in \text{DOMAIN } \text{transactions} \\
& \quad \wedge \text{mastership.master} = n \\
& \quad \wedge \vee \text{ReconcileChange}(n, i) \\
& \quad \quad \vee \text{ReconcileRollback}(n, i) \\
& \quad \wedge \text{UNCHANGED } \langle \text{mastership}, \text{conns} \rangle
\end{aligned}$$

$$\begin{aligned}
& \text{TypeOK} \triangleq \\
& \quad \forall i \in \text{DOMAIN } \text{transactions} : \\
& \quad \quad \wedge \text{transactions}[i].\text{index} \in \text{Nat} \\
& \quad \quad \wedge \text{transactions}[i].\text{phase} \in \{\text{Change}, \text{Rollback}\} \\
& \quad \quad \wedge \text{transactions}[i].\text{change.commit} \in \text{Status} \\
& \quad \quad \wedge \text{transactions}[i].\text{change.apply} \in \text{Status} \\
& \quad \quad \wedge \forall p \in \text{DOMAIN } \text{transactions}[i].\text{change.values} : \\
& \quad \quad \quad \text{transactions}[i].\text{change.values}[p] \neq \text{Nil} \Rightarrow \\
& \quad \quad \quad \text{transactions}[i].\text{change.values}[p] \in \text{STRING} \\
& \quad \quad \wedge \text{transactions}[i].\text{rollback.commit} \neq \text{Nil} \Rightarrow \\
& \quad \quad \quad \text{transactions}[i].\text{rollback.commit} \in \text{Status} \\
& \quad \quad \wedge \text{transactions}[i].\text{rollback.apply} \neq \text{Nil} \Rightarrow
\end{aligned}$$

$$\begin{aligned}
& \text{transactions}[i].\text{rollback.apply} \in \text{Status} \\
& \wedge \forall p \in \text{DOMAIN } \text{transactions}[i].\text{rollback.values} : \\
& \quad \text{transactions}[i].\text{rollback.values}[p] \neq \text{Nil} \Rightarrow \\
& \quad \text{transactions}[i].\text{rollback.values}[p] \in \text{STRING} \\
\text{LOCAL } \text{State} & \triangleq [\\
& \quad \text{transactions} \mapsto \text{transactions}, \\
& \quad \text{configuration} \mapsto \text{configuration}, \\
& \quad \text{mastership} \mapsto \text{mastership}, \\
& \quad \text{conns} \mapsto \text{conns}, \\
& \quad \text{target} \mapsto \text{target}] \\
\text{LOCAL } \text{Transitions} & \triangleq \\
& \text{LET} \\
& \quad \text{indexes} \triangleq \{i \in \text{DOMAIN } \text{transactions}' : \\
& \quad \quad i \in \text{DOMAIN } \text{transactions} \Rightarrow \text{transactions}'[i] \neq \text{transactions}[i]\} \\
& \text{IN} \quad [\text{transactions} \mapsto [i \in \text{indexes} \mapsto \text{transactions}'[i]]] @@ \\
& \quad (\text{IF } \text{configuration}' \neq \text{configuration} \text{ THEN } [\text{configuration} \mapsto \text{configuration}'] \text{ ELSE } \text{Empty}) @@ \\
& \quad (\text{IF } \text{target}' \neq \text{target} \text{ THEN } [\text{target} \mapsto \text{target}'] \text{ ELSE } \text{Empty}) @@ \\
& \quad (\text{IF } \text{Len}(\text{history}') > \text{Len}(\text{history}) \text{ THEN } [\text{event} \mapsto \text{history}'[\text{Len}(\text{history}')]] \text{ ELSE } \text{Empty}) \\
\text{Test} & \triangleq \text{INSTANCE } \text{Test} \text{ WITH} \\
& \quad \text{File} \leftarrow \text{"Transaction.log"}
\end{aligned}$$
