MODULE *Transaction*

INSTANCE *Naturals*

INSTANCE *FiniteSets*

INSTANCE *Sequences*

INSTANCE *TLC*

An empty constant
CONSTANT *Nil*

Transaction type constants
CONSTANTS
Change,
Rollback

$Type \triangleq \{Change, Rollback\}$

Proposal phase constants
CONSTANTS
Commit,
Apply

Status constants
CONSTANTS
Pending,
InProgress,
Complete,
Aborted,
Failed

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

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

The set of all nodes
CONSTANT *Node*

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

Variables defined by other modules.
VARIABLES
proposal,
configuration,

mastership,
target,
history

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

VARIABLE *transaction*

$$
\begin{array}{l}
\mathit{TypeOK} \triangleq \\
\quad \forall\, i \in \text{DOMAIN } \mathit{transaction} : \\
\qquad \land\, \mathit{transaction}[i].\mathit{type} \in \mathit{Type} \\
\qquad \land\, \mathit{transaction}[i].\mathit{index} \in \mathit{Nat} \\
\qquad \land\, \mathit{transaction}[i].\mathit{init} \in \mathit{Status} \\
\qquad \land\, \mathit{transaction}[i].\mathit{commit} \in \mathit{Status} \\
\qquad \land\, \mathit{transaction}[i].\mathit{apply} \in \mathit{Status} \\
\qquad \land\, \forall\, p \in \text{DOMAIN } \mathit{transaction}[i].\mathit{values} : \\
\qquad\quad \mathit{transaction}[i].\mathit{values}[p] \neq \mathit{Nil} \Rightarrow \mathit{transaction}[i].\mathit{values}[p] \in \text{STRING}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Test} \triangleq \text{INSTANCE } \mathit{Test} \text{ WITH} \\
\quad \mathit{File} \quad\quad \leftarrow \text{“Transaction.log”}, \\
\quad \mathit{CurrState} \leftarrow [\\
\qquad \mathit{transactions} \mapsto \mathit{transaction}, \\
\qquad \mathit{proposals} \quad\; \mapsto \mathit{proposal}, \\
\qquad \mathit{configuration} \mapsto \mathit{configuration}, \\
\qquad \mathit{mastership} \quad\; \mapsto \mathit{mastership}, \\
\qquad \mathit{target} \quad\quad\; \mapsto \mathit{target}], \\
\quad \mathit{SuccState} \leftarrow [\\
\qquad \mathit{transactions} \mapsto \mathit{transaction}', \\
\qquad \mathit{proposals} \quad\; \mapsto \mathit{proposal}', \\
\qquad \mathit{configuration} \mapsto \mathit{configuration}', \\
\qquad \mathit{mastership} \quad\; \mapsto \mathit{mastership}', \\
\qquad \mathit{target} \quad\quad\; \mapsto \mathit{target}']
\end{array}
$$

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

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

$$
\begin{array}{l}
\mathit{RequestChange}(p,\, v) \triangleq \\
\quad \land\, \mathit{transaction}' = \mathit{Append}(\mathit{transaction},\, [\\
\qquad\qquad\qquad \mathit{type} \quad\; \mapsto \mathit{Change}, \\
\qquad\qquad\qquad \mathit{index} \quad\; \mapsto 0, \\
\qquad\qquad\qquad \mathit{values} \quad\; \mapsto (p :> v), \\
\qquad\qquad\qquad \mathit{init} \quad\; \mapsto \mathit{InProgress}, \\
\qquad\qquad\qquad \mathit{commit} \mapsto \mathit{Pending}, \\
\qquad\qquad\qquad \mathit{apply} \quad\; \mapsto \mathit{Pending}])
\end{array}
$$

$\land$ UNCHANGED $\langle proposal,\ configuration,\ mastership,\ target,\ history\rangle$

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

$RequestRollback(i) \triangleq$
$\land\ transaction' = Append(transaction, [$
$type \mapsto Rollback,$
$index \mapsto i,$
$values \mapsto Empty,$
$init \mapsto InProgress,$
$commit \mapsto Pending,$
$apply \mapsto Pending])$
$\land$ UNCHANGED $\langle proposal,\ configuration,\ mastership,\ target,\ history\rangle$

This section models the *Transaction* log reconciler.

LOCAL $IsInitialized(i) \triangleq$
$i \in$ DOMAIN $transaction \Rightarrow transaction[i].init \in Done$

LOCAL $IsCommitted(i) \triangleq$
$i \in$ DOMAIN $transaction \Rightarrow transaction[i].commit \in Done$

LOCAL $IsApplied(i) \triangleq$
$i \in$ DOMAIN $transaction \Rightarrow transaction[i].apply \in Done$

$InitChange(n,\ i) \triangleq$
$\land\ \lor\ \land\ transaction[i].init = InProgress$

If the prior transaction has been initialized, initialize the transaction by appending the proposal and updating the transaction index.

$\land\ IsInitialized(i-1)$
$\land\ proposal' = Append(proposal, [$
$change \mapsto [$
$phase \mapsto Commit,$
$state \mapsto Pending,$
$values \mapsto [$
$p \in$ DOMAIN $transaction[i].values \mapsto [$
$index \mapsto Len(proposal)+1,$
$value \mapsto transaction[i].values[p]]]],$
$rollback \mapsto [$
$phase \mapsto Nil,$
$state \mapsto Nil,$
$revision \mapsto 0,$
$values \mapsto Empty]])$
$\land\ transaction' = [transaction$ EXCEPT $![i].index = Len(proposal'),$
$![i].init = Complete]$

$CommitChange(n,\ i) \triangleq$

$\land \lor \land\ transaction[i].commit = Pending$

$\land\ transaction[i].init = Complete$

A transaction cannot be committed until the prior transaction has been committed.

$\land\ IsCommitted(i-1)$

$\land\ transaction' = [transaction$ EXCEPT $![i].commit = InProgress]$

$\land$ UNCHANGED $\langle proposal \rangle$

$\lor \land\ transaction[i].commit = InProgress$

$\land\ proposal[transaction[i].index].change.phase = Commit$

If the change commit is still in the *Pending* state, set it to *InProgress*.

$\land \lor \land\ proposal[transaction[i].index].change.state = Pending$

$\land\ proposal' = [proposal$ EXCEPT $![transaction[i].index].change.state = InProgress,$

$![transaction[i].index].rollback.revision = configuration.commit$

$![transaction[i].index].rollback.values = [$

$p \in$ DOMAIN $proposal[transaction[i].index].change.values \mapsto$

IF $p \in$ DOMAIN $configuration.committed.values$ THEN

$configuration.committed.values[p]$

ELSE

$[index \mapsto 0,\ value \mapsto Nil]]]$

$\land$ UNCHANGED $\langle transaction \rangle$

If the change commit is *Complete*, mark the transaction *Complete*.

$\lor \land\ proposal[transaction[i].index].change.state = Complete$

$\land\ transaction' = [transaction$ EXCEPT $![i].commit = Complete]$

$\land$ UNCHANGED $\langle proposal \rangle$

If the change commit *Failed*, mark the transaction *Failed*.

$\lor \land\ proposal[transaction[i].index].change.state = Failed$

$\land\ transaction' = [transaction$ EXCEPT $![i].commit = Failed]$

$\land$ UNCHANGED $\langle proposal \rangle$

$ApplyChange(n,\ i) \triangleq$

$\land \lor \land\ transaction[i].apply = Pending$

If the commit phase was completed successfully, start the apply phase.

$\land \lor \land\ transaction[i].commit = Complete$

If the proposal is not in the apply phase, update the proposal phase.

$\land \lor \land\ proposal[transaction[i].index].change.phase \neq Apply$

$\land\ proposal' = [proposal$ EXCEPT $![transaction[i].index].change.phase = Apply,$

$![transaction[i].index].change.state = Pending]$

$\land$ UNCHANGED $\langle transaction \rangle$

If the proposal is in the apply phase and the previous transaction has completed the apply phase, start applying the change.

$\lor \land\ proposal[transaction[i].index].change.phase = Apply$

$\land\ proposal[transaction[i].index].change.state = Pending$

A transaction cannot be applied until the prior transaction has been applied.

$\land\ IsApplied(i-1)$

```
                    If the prior change failed being applied, it must be rolled back before
                    new changes can be applied.
                  ∧ ∧ transaction[i].index − 1 ∈ DOMAIN proposal
                    ∧ proposal[transaction[i].index − 1].change.phase = Apply
                    ∧ proposal[transaction[i].index − 1].change.state  = Failed
                    ⇒ ∧ proposal[transaction[i].index − 1].rollback.phase = Apply
                      ∧ proposal[transaction[i].index − 1].rollback.state  = Complete
                  ∧ transaction′ = [transaction EXCEPT ![i].apply = InProgress]
                  ∧ UNCHANGED ⟨proposal⟩
              If the commit phase was aborted or failed, abort the apply phase once the previous
              transaction has completed the apply phase.
            ∨ ∧ transaction[i].commit ∈ {Aborted, Failed}
                A transaction cannot be applied until the prior transaction has been applied.
              ∧ IsApplied(i − 1)
                If the prior change failed being applied, it must be rolled back before
                new changes can be applied.
              ∧ ∧ transaction[i].index − 1 ∈ DOMAIN proposal
                ∧ proposal[transaction[i].index − 1].change.phase = Apply
                ∧ proposal[transaction[i].index − 1].change.state  = Failed
                ⇒ ∧ proposal[transaction[i].index − 1].rollback.phase = Apply
                  ∧ proposal[transaction[i].index − 1].rollback.state  = Complete
              ∧ transaction′ = [transaction EXCEPT ![i].apply = Aborted]
              ∧ UNCHANGED ⟨proposal⟩
          ∨ ∧ transaction[i].apply = InProgress
            ∧ proposal[transaction[i].index].change.phase = Apply
                If the change apply is still in the Pending state, set it to InProgress.
            ∧ ∨ ∧ proposal[transaction[i].index].change.state = Pending
                ∧ proposal′ = [proposal EXCEPT ![transaction[i].index].change.state = InProgress]
                ∧ UNCHANGED ⟨transaction⟩
                If the change apply is Complete, mark the transaction Complete.
              ∨ ∧ proposal[transaction[i].index].change.state = Complete
                ∧ transaction′ = [transaction EXCEPT ![i].apply = Complete]
                ∧ UNCHANGED ⟨proposal⟩
                If the change apply Failed, mark the transaction Failed.
              ∨ ∧ proposal[transaction[i].index].change.state = Failed
                ∧ transaction′ = [transaction EXCEPT ![i].apply = Failed]
                ∧ UNCHANGED ⟨proposal⟩

ReconcileChange(n, i) ≜
  ∧ transaction[i].type = Change
  ∧ ∨ InitChange(n, i)
    ∨ CommitChange(n, i)
    ∨ ApplyChange(n, i)
```

$InitRollback(n,\ i) \triangleq$

$\land\ \lor\ \land\ transaction[i].init = InProgress$

Rollbacks cannot be initialized until all prior transactions have been initialized.

$\land\ IsInitialized(i-1)$

Rollback transactions must target valid proposal index.

$\land\ \lor\ \land\ transaction[i].index \in \text{DOMAIN } proposal$

To roll back a transaction, all subsequent transactions must be rolled back first.

Check whether the following proposal is being rolled back.

$\land\ \lor\ \land\ transaction[i].index + 1 \in \text{DOMAIN } proposal \Rightarrow$
$proposal[transaction[i].index + 1].rollback.phase \neq Nil$
$\land\ transaction' = [transaction \text{ EXCEPT } ![i].init = Complete]$

If the subsequent proposal is not being rolled back, fail the rollback transaction.

$\lor\ \land\ transaction[i].index + 1 \in \text{DOMAIN } proposal$
$\land\ proposal[transaction[i].index + 1].rollback.phase = Nil$
$\land\ transaction' = [transaction \text{ EXCEPT } ![i].init = Failed]$

If the rollback index is not a valid proposal index, fail the rollback request.

$\lor\ \land\ transaction[i].index \notin \text{DOMAIN } proposal$
$\land\ transaction' = [transaction \text{ EXCEPT } ![i].init = Failed]$

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

$CommitRollback(n,\ i) \triangleq$

$\land\ \lor\ \land\ transaction[i].commit = Pending$

A transaction cannot be committed until the prior transaction has been committed.

In the case of rollbacks, we serialize all state changes to ensure consistency
when rolling back changes.

$\land\ IsCommitted(i-1)$

If the transaction was initialized successfully, commit the rollback.

$\land\ \lor\ \land\ transaction[i].init = Complete$

If the target proposal is not yet being rolled back, transition the proposal.

$\land\ \lor\ \land\ proposal[transaction[i].index].rollback.phase = Nil$

Update the proposal's rollback state based on its change state.

$\land\ \lor\ \land\ proposal[transaction[i].index].change.phase = Commit$

If the target change is still pending, abort the change and rollback.

$\land\ \lor\ \land\ proposal[transaction[i].index].change.state = Pending$
$\land\ proposal' = [proposal \text{ EXCEPT } ![transaction[i].index].change.state = Aborted,$
$![transaction[i].index].rollback.phase = Commit$
$![transaction[i].index].rollback.state = Aborted]$
$\land \text{ UNCHANGED } \langle transaction \rangle$

If the target change is complete, start the rollback commit phase.

$\lor\ \land\ proposal[transaction[i].index].change.state = Complete$
$\land\ proposal' = [proposal \text{ EXCEPT } ![transaction[i].index].rollback.phase = Commit,$
$![transaction[i].index].rollback.state = Pending]$
$\land \text{ UNCHANGED } \langle transaction \rangle$

If the target change failed commit, complete the rollback commit.

$\lor \land proposal[transaction[i].index].change.state \in \{Aborted,\ Failed\}$
$\land transaction' = [transaction \text{ EXCEPT } ![i].commit = Complete]$
$\land \text{UNCHANGED } \langle proposal \rangle$

If the target change is in the *Apply* phasee, commit the rollback.

$\lor \land proposal[transaction[i].index].change.phase = Apply$
$\land proposal' = [proposal \text{ EXCEPT } ![transaction[i].index].rollback.phase = Commit,$
$\quad ![transaction[i].index].rollback.state = Pending]$
$\land \text{UNCHANGED } \langle transaction \rangle$

If the target rollback is being committed, transition the underlying proposal.

$\lor \land proposal[transaction[i].index].rollback.phase = Commit$

If the target proposal is being rolled back, begin the rollback commit
once the prior transaction has completed the commit phase.

$\land \lor \land proposal[transaction[i].index].rollback.state = Pending$
$\land transaction' = [transaction \text{ EXCEPT } ![i].commit = InProgress]$
$\land \text{UNCHANGED } \langle proposal \rangle$

If the target rollback was aborted, abort the transaction rollback
once the prior transaction has completed the commit phase.

$\lor \land proposal[transaction[i].index].rollback.state = Aborted$
$\land transaction' = [transaction \text{ EXCEPT } ![i].commit = Aborted]$
$\land \text{UNCHANGED } \langle proposal \rangle$

If the transaction failed initialization, abort the commit phase.

$\lor \land transaction[i].init \in \{Aborted,\ Failed\}$
$\land transaction' = [transaction \text{ EXCEPT } ![i].commit = Aborted]$
$\land \text{UNCHANGED } \langle proposal \rangle$

$\lor \land transaction[i].commit = InProgress$
$\land proposal[transaction[i].index].rollback.phase = Commit$

If the rollback commit is still in the *Pending* state, set it to *InProgress*.

$\land \lor \land proposal[transaction[i].index].rollback.state = Pending$
$\land proposal' = [proposal \text{ EXCEPT } ![transaction[i].index].rollback.state = InProgress]$
$\land \text{UNCHANGED } \langle transaction \rangle$

If the rollback commit is *Complete*, mark the transaction *Complete*.

$\lor \land proposal[transaction[i].index].rollback.state = Complete$
$\land transaction' = [transaction \text{ EXCEPT } ![i].commit = Complete]$
$\land \text{UNCHANGED } \langle proposal \rangle$

If the rollback commit *Failed*, mark the transaction *Failed*.

$\lor \land proposal[transaction[i].index].rollback.state = Failed$
$\land transaction' = [transaction \text{ EXCEPT } ![i].commit = Failed]$
$\land \text{UNCHANGED } \langle proposal \rangle$

$ApplyRollback(n,\ i) \triangleq$
$\land \lor \land transaction[i].apply = Pending$

A transaction cannot be applied until the prior transaction has been applied.
In the case of rollbacks, we serialize all state changes to ensure consistency
when rolling back changes.

$\land$ $IsApplied(i-1)$

If the commit phase was completed successfully, start the apply phase.

$\land$ $\lor$ $\land$ $transaction[i].commit = Complete$

If the target rollback is not yet being applied, transition the rollback.

$\land$ $\lor$ $\land$ $proposal[transaction[i].index].rollback.phase = Commit$

Update the proposal's rollback state based on its change state.

$\land$ $\lor$ $\land$ $proposal[transaction[i].index].change.phase = Apply$

If the target change is still pending, abort the change and rollback.

$\land$ $\lor$ $\land$ $proposal[transaction[i].index].change.state = Pending$
$\land$ $proposal' = [proposal$ EXCEPT $![transaction[i].index].change.state = Aborted,$
$![transaction[i].index].rollback.phase = Apply,$
$![transaction[i].index].rollback.state = Aborted]$
$\land$ UNCHANGED $\langle transaction \rangle$

If the target change is complete, start the rollback apply phase.

$\lor$ $\land$ $proposal[transaction[i].index].change.state = Complete$
$\land$ $proposal' = [proposal$ EXCEPT $![transaction[i].index].rollback.phase = Apply,$
$![transaction[i].index].rollback.state = Pending]$
$\land$ UNCHANGED $\langle transaction \rangle$

If the target change failed apply, complete the rollback apply.

$\lor$ $\land$ $proposal[transaction[i].index].change.state \in \{Aborted, Failed\}$
$\land$ $transaction' = [transaction$ EXCEPT $![i].apply = Complete]$
$\land$ UNCHANGED $\langle proposal \rangle$

If the target change is in the *Commit* phase, abort the change and rollback.

$\lor$ $\land$ $proposal[transaction[i].index].change.phase = Commit$
$\land$ $proposal' = [proposal$ EXCEPT $![transaction[i].index].change.state = Aborted,$
$![transaction[i].index].rollback.phase = Apply,$
$![transaction[i].index].rollback.state = Aborted]$
$\land$ UNCHANGED $\langle transaction \rangle$

If the target rollback is being applied, transition the underlying proposal.

$\lor$ $\land$ $proposal[transaction[i].index].rollback.phase = Apply$

If the target proposal is being rolled back, begin the rollback apply
once the prior transaction has completed the apply phase.

$\land$ $\lor$ $\land$ $proposal[transaction[i].index].rollback.state = Pending$
$\land$ $transaction' = [transaction$ EXCEPT $![i].apply = InProgress]$
$\land$ UNCHANGED $\langle proposal \rangle$

If the target rollback was aborted, abort the transaction rollback
once the prior transaction has completed the apply phase.

$\lor$ $\land$ $proposal[transaction[i].index].rollback.state = Aborted$
$\land$ $transaction' = [transaction$ EXCEPT $![i].apply = Aborted]$
$\land$ UNCHANGED $\langle proposal \rangle$

If the transaction failed initialization, abort the apply phase.

$\lor$ $\land$ $transaction[i].init \in \{Aborted, Failed\}$
$\land$ $transaction' = [transaction$ EXCEPT $![i].apply = Aborted]$
$\land$ UNCHANGED $\langle proposal \rangle$

$\lor$ $\land$ $transaction[i].apply = InProgress$

```
        ∧ proposal[transaction[i].index].rollback.phase = Apply
            If the rollback apply is still in the Pending state, set it to InProgress.
        ∧ ∨ ∧ proposal[transaction[i].index].rollback.state = Pending
            ∧ proposal' = [proposal EXCEPT ![transaction[i].index].rollback.state = InProgress]
            ∧ UNCHANGED ⟨transaction⟩
            If the rollback apply is Complete, mark the transaction Complete.
          ∨ ∧ proposal[transaction[i].index].rollback.state    = Complete
            ∧ transaction' = [transaction EXCEPT ![i].apply = Complete]
            ∧ UNCHANGED ⟨proposal⟩
            If the rollback apply Failed, mark the transaction Failed.
          ∨ ∧ proposal[transaction[i].index].rollback.state    = Failed
            ∧ transaction' = [transaction EXCEPT ![i].apply = Failed]
            ∧ UNCHANGED ⟨proposal⟩

ReconcileRollback(n, i) ≜
    ∧ transaction[i].type = Rollback
    ∧ ∨ InitRollback(n, i)
      ∨ CommitRollback(n, i)
      ∨ ApplyRollback(n, i)

 Reconcile a transaction
ReconcileTransaction(n, i) ≜
    ∧ i ∈ DOMAIN transaction
    ∧ ∨ ReconcileChange(n, i)
      ∨ ReconcileRollback(n, i)
    ∧ UNCHANGED ⟨configuration, mastership, target, history⟩
```