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

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

VARIABLE *transaction*

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

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

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

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

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

$$
\begin{array}{l}
RequestRollback(i) \triangleq \\
\quad \land\ transaction' = Append(transaction, [\\
\qquad\qquad\qquad type \mapsto Rollback, \\
\qquad\qquad\qquad index \mapsto i, \\
\qquad\qquad\qquad values \mapsto Empty, \\
\qquad\qquad\qquad init \mapsto InProgress, \\
\qquad\qquad\qquad commit \mapsto Pending, \\
\qquad\qquad\qquad apply \mapsto Pending]) \\
\quad \land\ \text{UNCHANGED}\ \langle proposal, configuration, mastership, target \rangle
\end{array}
$$

This section models the *Transaction* log reconciler.

$$
\begin{array}{l}
\text{LOCAL}\ IsInitialized(i) \triangleq \\
\quad i \in \text{DOMAIN}\ transaction \Rightarrow transaction[i].init \in Done
\end{array}
$$

$$
\begin{array}{l}
\text{LOCAL}\ IsCommitted(i) \triangleq \\
\quad i \in \text{DOMAIN}\ transaction \Rightarrow transaction[i].commit \in Done
\end{array}
$$

$$
\begin{array}{l}
\text{LOCAL}\ IsApplied(i) \triangleq \\
\quad i \in \text{DOMAIN}\ transaction \Rightarrow transaction[i].apply \in Done
\end{array}
$$

$$
\begin{array}{l}
InitChange(n, i) \triangleq \\
\quad \land\ \lor\ \land\ transaction[i].init = InProgress \\
\qquad\qquad \text{If the prior transaction has been initialized, initialize the transaction by} \\
\qquad\qquad \text{appending the proposal and updating the transaction index.} \\
\qquad\quad \land\ IsInitialized(i-1) \\
\qquad\quad \land\ proposal' = Append(proposal, [\\
\qquad\qquad\qquad\qquad change \mapsto [\\
\qquad\qquad\qquad\qquad\quad phase \mapsto Commit, \\
\qquad\qquad\qquad\qquad\quad state \mapsto Pending, \\
\qquad\qquad\qquad\qquad\quad values \mapsto transaction[i].change], \\
\qquad\qquad\qquad\qquad rollback \mapsto [\\
\qquad\qquad\qquad\qquad\quad phase \mapsto Nil, \\
\qquad\qquad\qquad\qquad\quad state \mapsto Nil, \\
\qquad\qquad\qquad\qquad\quad revision \mapsto 0, \\
\qquad\qquad\qquad\qquad\quad values \mapsto Empty]]) \\
\qquad\quad \land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].index = Len(proposal'), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![i].init = Complete]
\end{array}
$$

$$
\begin{array}{l}
CommitChange(n, i) \triangleq \\
\quad \land\ \lor\ \land\ transaction[i].commit = Pending \\
\qquad\quad \land\ transaction[i].init = Complete
\end{array}
$$

```
        A transaction cannot be committed until the prior transaction has been committed.
      ∧ IsCommitted(i − 1)
      ∧ transaction′ = [transaction EXCEPT ![i].commit = InProgress]
    ∨ ∧ transaction[i].commit = InProgress
      ∧ proposal[transaction[i].index].change.phase = Commit
          If the change commit 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,
                                         ![transaction[i].index].rollback.revision = configuration.commit
                                         ![transaction[i].index].rollback.values   = [
                                            p ∈ DOMAIN proposal[transaction[i].index].change.values ↦
                                              IF p ∈ DOMAIN configuration.committed.values THEN
                                                 configuration.committed.values[p]
                                               ELSE
                                                 [index ↦ 0, value ↦ Nil]]]
          ∧ UNCHANGED ⟨transaction⟩
        If the change commit is Complete, mark the transaction Complete.
        ∨ ∧ proposal[transaction[i].index].change.state = Complete
          ∧ transaction′ = [transaction EXCEPT ![i].commit = Complete]
          ∧ UNCHANGED ⟨proposal⟩
        If the change commit Failed, mark the transaction Failed.
        ∨ ∧ proposal[transaction[i].index].change.state = Failed
          ∧ transaction′ = [transaction EXCEPT ![i].commit = Failed]
          ∧ UNCHANGED ⟨proposal⟩

ApplyChange(n, i) ≜
  ∧ ∨ ∧ transaction[i].apply = Pending
        If the commit phase was completed successfully, start the apply phase.
    ∧ ∨ ∧ transaction[i].commit = Complete
          If the proposal is not in the apply phase, update the proposal phase.
      ∧ ∨ ∧ proposal[transaction[i].index].change.phase ≠ Apply
          ∧ proposal′ = [proposal EXCEPT ![transaction[i].index].change.phase = Apply,
                                         ![transaction[i].index].change.state  = Pending]
          ∧ UNCHANGED ⟨transaction⟩
        If the proposal is in the apply phase and the previous transaction has completed
        the apply phase, start applying the change.
        ∨ ∧ proposal[transaction[i].index].change.phase = Apply
          ∧ proposal[transaction[i].index].change.state  = Pending
            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
```

$\land\ proposal[transaction[i].index - 1].change.state\ = Failed$
$\Rightarrow\ \land\ proposal[transaction[i].index - 1].rollback.phase = Apply$
$\land\ proposal[transaction[i].index - 1].rollback.state\ = Complete$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].apply = InProgress]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

If the commit phase was aborted or failed, abort the apply phase once the previous transaction has completed the apply phase.

$\lor\ \land\ transaction[i].commit \in \{Aborted,\ Failed\}$

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

$\land\ IsApplied(i - 1)$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].apply = Aborted]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

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

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

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

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

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

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

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

$ReconcileChange(n,\ i)\ \triangleq$
$\land\ transaction[i].type = Change$
$\land\ \lor\ InitChange(n,\ i)$
$\lor\ CommitChange(n,\ i)$
$\lor\ ApplyChange(n,\ i)$

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

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.
              ∨ ∧ transaction[i].index + 1 ∈ DOMAIN proposal
                ∧ proposal[transaction[i].index + 1].rollback.phase = Nil
                ∧ transaction' = [transaction EXCEPT ![i].init = Failed]
            If the rollback index is not a valid proposal index, fail the rollback request.
          ∨ ∧ transaction[i].index ∉ DOMAIN proposal
            ∧ transaction' = [transaction EXCEPT ![i].init = Failed]
    ∧ UNCHANGED ⟨proposal⟩

CommitRollback(n, i) ≜
    ∧ ∨ ∧ 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.
      ∧ IsCommitted(i − 1)
          If the transaction was initialized successfully, commit the rollback.
      ∧ ∨ ∧ transaction[i].init = Complete
                If the target proposal is not yet being rolled back, transition the proposal.
            ∧ ∨ ∧ proposal[transaction[i].index].rollback.phase = Nil
                      Update the proposal's rollback state based on its change state.
                  ∧ ∨ ∧ proposal[transaction[i].index].change.phase = Commit
                            If the target change is still pending, abort the change and rollback.
                        ∧ ∨ ∧ proposal[transaction[i].index].change.state = Pending
                            ∧ proposal' = [proposal EXCEPT ![transaction[i].index].change.state   = Aborted,
                                                           ![transaction[i].index].rollback.phase = Commit
                                                           ![transaction[i].index].rollback.state  = Aborted]
                            ∧ UNCHANGED ⟨transaction⟩
                          If the target change is complete, start the rollback commit phase.
                        ∨ ∧ proposal[transaction[i].index].change.state = Complete
                            ∧ proposal' = [proposal EXCEPT ![transaction[i].index].rollback.phase = Commit,
                                                           ![transaction[i].index].rollback.state  = Pending]
                            ∧ UNCHANGED ⟨transaction⟩
                          If the target change failed commit, complete the rollback commit.
                        ∨ ∧ proposal[transaction[i].index].change.state ∈ {Aborted, Failed}
                            ∧ transaction' = [transaction EXCEPT ![i].commit = Complete]
                            ∧ UNCHANGED ⟨proposal⟩
                      If the target change is in the Apply phasee, commit the rollback.
                    ∨ ∧ proposal[transaction[i].index].change.phase = Apply
                      ∧ proposal' = [proposal EXCEPT ![transaction[i].index].rollback.phase = Commit,
                                                     ![transaction[i].index].rollback.state  = Pending]
                      ∧ UNCHANGED ⟨transaction⟩
                If the target rollback is being committed, transition the underlying proposal.
              ∨ ∧ 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.
            ∧ ∨ ∧ proposal[transaction[i].index].rollback.state = Pending
                ∧ transaction' = [transaction EXCEPT ![i].commit = InProgress]
                ∧ UNCHANGED ⟨proposal⟩
              If the target rollback was aborted, abort the transaction rollback
              once the prior transaction has completed the commit phase.
              ∨ ∧ proposal[transaction[i].index].rollback.state = Aborted
                ∧ transaction' = [transaction EXCEPT ![i].commit = Aborted]
                ∧ UNCHANGED ⟨proposal⟩
        If the transaction failed initialization, abort the commit phase.
        ∨ ∧ transaction[i].init ∈ {Aborted, Failed}
          ∧ transaction' = [transaction EXCEPT ![i].commit = Aborted]
          ∧ UNCHANGED ⟨proposal⟩
    ∨ ∧ transaction[i].commit = InProgress
      ∧ proposal[transaction[i].index].rollback.phase = Commit
        If the rollback commit 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 commit is Complete, mark the transaction Complete.
        ∨ ∧ proposal[transaction[i].index].rollback.state = Complete
          ∧ transaction' = [transaction EXCEPT ![i].commit = Complete]
          ∧ UNCHANGED ⟨proposal⟩
        If the rollback commit Failed, mark the transaction Failed.
        ∨ ∧ proposal[transaction[i].index].rollback.state = Failed
          ∧ transaction' = [transaction EXCEPT ![i].commit = Failed]
          ∧ UNCHANGED ⟨proposal⟩

ApplyRollback(n, i) ≜
  ∧ ∨ ∧ 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.
    ∧ IsApplied(i − 1)
      If the commit phase was completed successfully, start the apply phase.
    ∧ ∨ ∧ transaction[i].commit = Complete
          If the target rollback is not yet being applied, transition the rollback.
        ∧ ∨ ∧ proposal[transaction[i].index].rollback.phase = Commit
              Update the proposal's rollback state based on its change state.
            ∧ ∨ ∧ proposal[transaction[i].index].change.phase = Apply
                  If the target change is still pending, abort the change and rollback.
                ∧ ∨ ∧ proposal[transaction[i].index].change.state = Pending
                    ∧ proposal' = [proposal EXCEPT ![transaction[i].index].change.state   = Aborted,
                                                   ![transaction[i].index].rollback.phase = Apply,
```

$$\quad ![transaction[i].index].rollback.state = Aborted]$$
$$\quad \wedge \text{UNCHANGED } \langle transaction \rangle$$
If the target change is complete, start the rollback apply phase.
$$\vee \wedge proposal[transaction[i].index].change.state = Complete$$
$$\quad \wedge proposal' = [proposal \text{ EXCEPT } ![transaction[i].index].rollback.phase = Apply,$$
$$\quad\quad ![transaction[i].index].rollback.state = Pending]$$
$$\quad \wedge \text{UNCHANGED } \langle transaction \rangle$$
If the target change failed apply, complete the rollback apply.
$$\vee \wedge proposal[transaction[i].index].change.state \in \{Aborted, Failed\}$$
$$\quad \wedge transaction' = [transaction \text{ EXCEPT } ![i].apply = Complete]$$
$$\quad \wedge \text{UNCHANGED } \langle proposal \rangle$$
If the target change is in the *Commit* phase, abort the change and rollback.
$$\vee \wedge proposal[transaction[i].index].change.phase = Commit$$
$$\quad \wedge proposal' = [proposal \text{ EXCEPT } ![transaction[i].index].change.state = Aborted,$$
$$\quad\quad ![transaction[i].index].rollback.phase = Apply,$$
$$\quad\quad ![transaction[i].index].rollback.state = Aborted]$$
$$\quad \wedge \text{UNCHANGED } \langle transaction \rangle$$
If the target rollback is being applied, transition the underlying proposal.
$$\vee \wedge proposal[transaction[i].index].rollback.phase = Apply$$
If the target proposal is being rolled back, begin the rollback apply
once the prior transaction has completed the apply phase.
$$\wedge \vee \wedge proposal[transaction[i].index].rollback.state = Pending$$
$$\quad \wedge transaction' = [transaction \text{ EXCEPT } ![i].apply = InProgress]$$
$$\quad \wedge \text{UNCHANGED } \langle proposal \rangle$$
If the target rollback was aborted, abort the transaction rollback
once the prior transaction has completed the apply phase.
$$\vee \wedge proposal[transaction[i].index].rollback.state = Aborted$$
$$\quad \wedge transaction' = [transaction \text{ EXCEPT } ![i].apply = Aborted]$$
$$\quad \wedge \text{UNCHANGED } \langle proposal \rangle$$
If the transaction failed initialization, abort the apply phase.
$$\vee \wedge transaction[i].init \in \{Aborted, Failed\}$$
$$\quad \wedge transaction' = [transaction \text{ EXCEPT } ![i].apply = Aborted]$$
$$\quad \wedge \text{UNCHANGED } \langle proposal \rangle$$
$$\vee \wedge transaction[i].apply = InProgress$$
$$\quad \wedge proposal[transaction[i].index].rollback.phase = Apply$$
If the rollback apply is still in the *Pending* state, set it to *InProgress*.
$$\wedge \vee \wedge proposal[transaction[i].index].rollback.state = Pending$$
$$\quad \wedge proposal' = [proposal \text{ EXCEPT } ![transaction[i].index].rollback.state = InProgress]$$
$$\quad \wedge \text{UNCHANGED } \langle transaction \rangle$$
If the rollback apply is *Complete*, mark the transaction *Complete*.
$$\vee \wedge proposal[transaction[i].index].rollback.state = Complete$$
$$\quad \wedge transaction' = [transaction \text{ EXCEPT } ![i].apply = Complete]$$
$$\quad \wedge \text{UNCHANGED } \langle proposal \rangle$$
If the rollback apply *Failed*, mark the transaction *Failed*.
$$\vee \wedge proposal[transaction[i].index].rollback.state = Failed$$

$\land\ transaction' = [transaction \text{ EXCEPT } ![i].apply = Failed]$
$\land \text{ UNCHANGED } \langle proposal \rangle$

$ReconcileRollback(n,\ i) \triangleq$
$\land\ transaction[i].type = Rollback$
$\land\ \lor\ InitRollback(n,\ i)$
$\lor\ CommitRollback(n,\ i)$
$\lor\ ApplyRollback(n,\ i)$

Reconcile a transaction
$ReconcileTransaction(n,\ i) \triangleq$
$\land\ i \in \text{DOMAIN } transaction$
$\land\ \lor\ ReconcileChange(n,\ i)$
$\lor\ ReconcileRollback(n,\ i)$
$\land \text{ UNCHANGED } \langle configuration,\ mastership,\ target \rangle$