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*

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

VARIABLE *transaction*

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

$$
\begin{array}{l}
\text{LOCAL } \mathit{State} \triangleq [ \\
\quad \mathit{transactions} \mapsto [i \in \text{DOMAIN } \mathit{transaction} \mapsto \mathit{transaction}[i] \mathbin{@@} [\mathit{index} \mapsto i]], \\
\quad \mathit{proposals} \mapsto [i \in \text{DOMAIN } \mathit{proposal} \mapsto \mathit{proposal}[i] \mathbin{@@} [\mathit{index} \mapsto i]], \\
\quad \mathit{configuration} \mapsto \mathit{configuration}]
\end{array}
$$

$$
\begin{array}{l}
\text{LOCAL } \mathit{Transitions} \triangleq \\
\quad \text{LET} \\
\qquad \mathit{transactions} \triangleq \{i \in \text{DOMAIN } \mathit{transaction}' : \\
\qquad\qquad\qquad i \in \text{DOMAIN } \mathit{transaction} \Rightarrow \mathit{transaction}'[i] \neq \mathit{transaction}[i]\} \\
\qquad \mathit{proposals} \triangleq \{i \in \text{DOMAIN } \mathit{proposal}' : \\
\qquad\qquad\qquad i \in \text{DOMAIN } \mathit{proposal} \Rightarrow \mathit{proposal}'[i] \neq \mathit{proposal}[i]\} \\
\quad \text{IN} \\
\qquad [\mathit{transactions} \mapsto [i \in \mathit{transactions} \mapsto \mathit{transaction}'[i] \mathbin{@@} [\mathit{index} \mapsto i]], \\
\qquad\ \mathit{proposals} \mapsto [i \in \mathit{proposals} \mapsto \mathit{proposal}'[i] \mathbin{@@} [\mathit{index} \mapsto i]]]
\end{array}
$$

$$
\mathit{Test} \triangleq \text{INSTANCE } \mathit{Test} \text{ WITH } \mathit{File} \leftarrow \text{“Transaction.log”}
$$

---

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

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

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

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

---

This section models the *Transaction* log reconciler.

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

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

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

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

$CommitChange(n,\ i)\ \triangleq$

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

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

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

$\land\ IsCommitted(i-1)$

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

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

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

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

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

$\land\ proposal' = [proposal\ \text{EXCEPT}\ ![transaction[i].proposal].change.commit = InProgress,$

$![transaction[i].proposal].rollback.revision = configuration.co$

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

$p \in \text{DOMAIN}\ proposal[transaction[i].proposal].change.values$

$\text{IF}\ p \in \text{DOMAIN}\ configuration.committed.values\ \text{THEN}$

$configuration.committed.values[p]$

$\text{ELSE}$

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

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

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

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

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

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

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

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

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

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

$ApplyChange(n,\ i)\ \triangleq$

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

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

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

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

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

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.

$\land\ \land\ transaction[i].proposal - 1 \in \text{DOMAIN}\ proposal$

$\land\ proposal[transaction[i].proposal\ - 1].change.apply = Failed$

$\Rightarrow proposal[transaction[i].proposal - 1].rollback.apply = Complete$

$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].apply = InProgress]$

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

```
                If the commit phase was aborted or failed, abort the apply phase once the previous
                transaction has completed the apply phase.
            ∨ ∧ 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].proposal − 1 ∈ DOMAIN proposal
                  ∧ proposal[transaction[i].proposal − 1].change.apply = Failed
                  ⇒ proposal[transaction[i].proposal − 1].rollback.apply = Complete
              ∧ transaction′ = [transaction EXCEPT ![i].apply = Aborted]
              ∧ UNCHANGED ⟨proposal⟩
        ∨ ∧ transaction[i].apply = InProgress
            If the change apply is still in the Pending state, set it to InProgress.
          ∧ ∨ ∧ proposal[transaction[i].proposal].change.apply = Pending
              ∧ proposal′ = [proposal EXCEPT ![transaction[i].proposal].change.apply = InProgress]
              ∧ UNCHANGED ⟨transaction⟩
            If the change apply is Complete, mark the transaction Complete.
            ∨ ∧ proposal[transaction[i].proposal].change.apply = Complete
              ∧ transaction′ = [transaction EXCEPT ![i].apply = Complete]
              ∧ UNCHANGED ⟨proposal⟩
            If the change apply Failed, mark the transaction Failed.
            ∨ ∧ proposal[transaction[i].proposal].change.apply = 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) ≜
  ∧ ∨ ∧ transaction[i].init = InProgress
        Rollbacks cannot be initialized until all prior transactions have been initialized.
      ∧ IsInitialized(i − 1)
        Rollback transactions must target valid proposal index.
      ∧ ∨ ∧ transaction[i].proposal ∈ DOMAIN proposal
            To roll back a transaction, all subsequent transactions must be rolled back first.
            Check whether the following proposal is being rolled back.
          ∧ ∨ ∧ transaction[i].proposal + 1 ∈ DOMAIN proposal ⇒
                    proposal[transaction[i].proposal + 1].phase = Rollback
              ∧ transaction′ = [transaction EXCEPT ![i].init = Complete]
            If the subsequent proposal is not being rolled back, fail the rollback transaction.
```

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

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

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

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

$CommitRollback(n,\ i) \triangleq$

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

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.

$\land \lor \land proposal[transaction[i].proposal].phase = Change$

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

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

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

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

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

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

If the target proposal is being rolled back, transition the underlying proposal.

$\lor \land proposal[transaction[i].proposal].phase = Rollback$

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

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

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

$\lor \land proposal[transaction[i].proposal].rollback.commit = Aborted$

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

If the transaction failed initialization, abort the commit phase.

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

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

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

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

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

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

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

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

$ApplyRollback(n,\ i)\ \triangleq$

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

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

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

If the target change has not yet been applied, abort the change and rollback.

$\land\ \lor\ \land\ \lor\ proposal[transaction[i].proposal].change.commit \notin Done$
$\lor\ proposal[transaction[i].proposal].change.apply = Pending$
$\land\ proposal' = [proposal\ \text{EXCEPT}\ ![transaction[i].proposal].change.apply = Aborted,$
$![transaction[i].proposal].rollback.apply = Aborted]$
$\land\ \text{UNCHANGED}\ \langle transaction \rangle$

If the target change has been applied, begin applying the rollback.

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

If the target change was aborted or failed, complete applying the rollback.

$\lor\ \land\ proposal[transaction[i].proposal].change.apply = Aborted$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].apply = Complete]$

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

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

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

If the transaction failed commit, abort the apply phase.

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

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

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

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

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

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

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

$\lor \land proposal[transaction[i].proposal].rollback.apply = Failed$
$\land transaction' = [transaction \text{ EXCEPT } ![i].apply = Failed]$
$\land$ 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

$ReconcileTransaction(n, i) \triangleq$
$\land i \in$ DOMAIN $transaction$
$\land \lor ReconcileChange(n, i)$
$\lor ReconcileRollback(n, i)$
$\land$ UNCHANGED $\langle configuration \rangle$