```
──────────────── MODULE Transaction ────────────────
INSTANCE Naturals

INSTANCE FiniteSets

INSTANCE Sequences

INSTANCE TLC
────────────────────────────────────────────────────

  An empty constant
CONSTANT Nil

  Transaction type constants
CONSTANTS
   Change,
   Rollback

  Transaction isolation constants
CONSTANTS
   ReadCommitted,
   Serializable

  Phase constants
CONSTANTS
   Initialize,
   Validate,
   Abort,
   Commit,
   Apply

Phase ≜
   {Initialize,
    Validate,
    Abort,
    Commit,
    Apply}

  Status constants
CONSTANTS
   InProgress,
   Complete,
   Failed

State ≜
   {InProgress,
    Complete,
```

$\quad\quad \mathit{Failed}\}$

State constants

CONSTANTS
$\quad \mathit{Pending},$
$\quad \mathit{Validated},$
$\quad \mathit{Committed},$
$\quad \mathit{Applied},$
$\quad \mathit{Aborted}$

$\mathit{Status} \triangleq$
$\quad \{\mathit{Pending},$
$\quad\ \ \mathit{Validated},$
$\quad\ \ \mathit{Committed},$
$\quad\ \ \mathit{Applied},$
$\quad\ \ \mathit{Aborted}\}$

CONSTANTS
$\quad \mathit{Valid},$
$\quad \mathit{Invalid}$

CONSTANTS
$\quad \mathit{Success},$
$\quad \mathit{Failure}$

The set of all nodes
CONSTANT Node

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

Example:
$\quad \mathit{Target} \triangleq$
$\quad\quad [\mathit{target}1 \mapsto$
$\quad\quad\quad [\text{persistent} \mapsto \text{FALSE}, \mathit{values} \mapsto [$
$\quad\quad\quad\quad \mathit{path}1 \mapsto \{\text{"}\mathit{value}1\text{"}, \text{"}\mathit{value}2\text{"}\},$
$\quad\quad\quad\quad \mathit{path}2 \mapsto \{\text{"}\mathit{value}2\text{"}, \text{"}\mathit{value}3\text{"}\}]],$
$\quad\quad \mathit{target}2 \mapsto$
$\quad\quad\quad [\text{persistent} \mapsto \text{TRUE}, \mathit{values} \mapsto [$
$\quad\quad\quad\quad \mathit{path}2 \mapsto \{\text{"}\mathit{value}3\text{"}, \text{"}\mathit{value}4\text{"}\},$
$\quad\quad\quad\quad \mathit{path}3 \mapsto \{\text{"}\mathit{value}4\text{"}, \text{"}\mathit{value}5\text{"}\}]]]$

CONSTANT Target

$\mathit{Empty} \triangleq [p \in \{\} \mapsto [\mathit{value} \mapsto \mathit{Nil}, \mathit{delete} \mapsto \text{FALSE}]]$

A transaction log. Transactions may either request a set
of changes to a set of targets or rollback a prior change.
VARIABLE $\mathit{transaction}$

A record of per-target proposals
```
VARIABLE $proposal$
```

A record of per-target configurations
```
VARIABLE $configuration$
```

A record of target states
```
VARIABLE $target$
```

A record of target masterships
```
VARIABLE $mastership$
```

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
```
$RequestChange(i, c) \triangleq$
    $\land i = Len(transaction) + 1$
    $\land \exists isolation \in \{ReadCommitted, Serializable\}$ :
        $\land transaction' = transaction$ @@ $(i :> [type \mapsto Change,$
                                                  $isolation \mapsto isolation,$
                                                  $change \mapsto c,$
                                                  $targets \mapsto \{\},$
                                                  $phase \mapsto Initialize,$
                                                  $state \mapsto InProgress,$
                                                  $status \mapsto Pending])$
    $\land$ UNCHANGED $\langle proposal, configuration, mastership, target \rangle$
```

Add a rollback of transaction 't' to the transaction log
```
$RequestRollback(i, j) \triangleq$
    $\land i = Len(transaction) + 1$
    $\land \exists isolation \in \{ReadCommitted, Serializable\}$ :
        $\land transaction' = transaction$ @@ $(i :> [type \mapsto Rollback,$
                                                  $isolation \mapsto isolation,$
                                                  $rollback \mapsto j,$
                                                  $targets \mapsto \{\},$
                                                  $phase \mapsto Initialize,$
                                                  $state \mapsto InProgress,$
                                                  $status \mapsto Pending])$
    $\land$ UNCHANGED $\langle proposal, configuration, mastership, target \rangle$
```

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

Transacations proceed through a series of phases:
* *Initialize* - create and link Proposals
* *Validate* - validate changes and rollbacks
* *Commit* - commit changes to Configurations
* *Apply* - commit changes to Targets

```
 Reconcile a transaction
$ReconcileTransaction(n, i) \triangleq$
        Initialize is the only transaction phase that's globally serialized.
        While in the Initializing phase, the reconciler checks whether the
        prior transaction has been Initialized before creating Proposals in
        the Initialize phase. Once all of the transaction's proposals have
        been Initialized, the transaction will be marked Initialized. If any
        proposal is Failed, the transaction will be marked Failed as well.
    $\land \lor \land transaction[i].phase = Initialize$
          $\land \lor \land transaction[i].state = InProgress$
                 All prior transaction must be initialized before proceeding
                 to initialize this transaction.
                $\land \neg \exists j \in \text{DOMAIN } transaction :$
                      $\land j < i$
                      $\land transaction[j].phase = Initialize$
                      $\land transaction[j].state \ = InProgress$
                   If the transaction's targets are not yet set, create proposals
                   and add targets to the transaction state.
                $\land \lor \land transaction[i].targets = \{\}$
                         If the transaction is a change, the targets are taken
                         from the change values.
                      $\land \lor \land transaction[i].type = Change$
                            $\land transaction' = [transaction \text{ EXCEPT } ![i].targets = \text{DOMAIN } transaction[i].change]$
                            $\land proposal' = [t \in \text{DOMAIN } proposal \mapsto$
                                 $\text{IF } t \in \text{DOMAIN } transaction[i].change \text{ THEN}$
                                    $proposal[t] @@ (i :> [type \mapsto Change,$
                                                           $change \mapsto$
                                                             $[index \mapsto i,$
                                                              $values \mapsto transaction[i].change[t]],$
                                                           $rollback \mapsto$
                                                             $[index \mapsto 0,$
                                                              $values \mapsto Empty],$
                                                           $dependency \mapsto [index \mapsto 0],$
                                                           $phase \mapsto Initialize,$
                                                           $state \mapsto InProgress])$
                                 $\text{ELSE}$
                                    $proposal[t]]$
        If the transaction is a rollback, the targets affected are
        the targets of the change transaction being rolled back.
        $\lor \land transaction[i].type = Rollback$
                If the rollback index is a valid Change transaction,
                initialize proposals for all of the Change targets.
           $\land \lor \land transaction[i].rollback \in \textsc{domain}\ transaction$
                   $\land transaction[transaction[i].rollback].type = Change$
                   $\land transaction' = [transaction\ \textsc{except}\ ![i].targets =$
                                     $\textsc{domain}\ transaction[transaction[i].rollback].change]$
                   $\land proposal' = [t \in \textsc{domain}\ proposal \mapsto$
                         $\textsc{if}\ t \in \textsc{domain}\ transaction[transaction[i].rollback].change\ \textsc{then}$
                            $proposal[t] @@ (i :> [type \mapsto Rollback,$
                                                   $change \mapsto$
                                                     $[index \mapsto 0,$
                                                      $values \mapsto Empty],$
                                                   $rollback \mapsto$
                                                     $[index \mapsto transaction[i].rollback,$
                                                      $values \mapsto Empty],$
                                                   $dependency \mapsto [index \mapsto 0],$
                                                   $phase \mapsto Initialize,$
                                                   $state \mapsto InProgress])$
                          $\textsc{else}$
                            $proposal[t]]$
                If the rollback index is not a valid Change transaction
                fail the Rollback transaction.
              $\lor \land \lor \land transaction[i].rollback \in \textsc{domain}\ transaction$
                         $\land transaction[transaction[i].rollback].type = Rollback$
                      $\lor transaction[i].rollback \notin \textsc{domain}\ transaction$
                   $\land transaction' = [transaction\ \textsc{except}\ ![i].state = Failed]$
                   $\land \textsc{unchanged}\ \langle proposal \rangle$
    If the transaction's proposals have been initialized, check proposals
    for completion or failures.
    $\lor \land transaction[i].targets \neq \{\}$
        If all proposals have been Complete, mark the transaction Complete.
       $\land \lor \land \forall t \in transaction[i].targets :$
                   $\land proposal[t][i].phase = Initialize$
                   $\land proposal[t][i].state = Complete$
             $\land transaction' = [transaction\ \textsc{except}\ ![i].state = Complete]$
             $\land \textsc{unchanged}\ \langle proposal \rangle$
        If any proposal has been Failed, mark the transaction Failed.
          $\lor \land \exists t \in transaction[i].targets :$
                   $\land proposal[t][i].phase = Initialize$
                   $\land proposal[t][i].state = Failed$
             $\land transaction' = [transaction\ \textsc{except}\ ![i].state = Failed]$
             $\land \textsc{unchanged}\ \langle proposal \rangle$
        Once the transaction has been Initialized, proceed to the *Validate* phase.
        If any of the transaction's proposals depend on a *Serializable* transaction,
        verify the dependency has been *Validated* to preserve serializability before
        moving the transaction to the *Validate* phase.
        $\lor \land transaction[i].state = Complete$
          $\land \forall t \in transaction[i].targets :$
                $\land proposal[t][i].dependency.index \in \text{DOMAIN } transaction$
                $\land transaction[proposal[t][i].dependency.index].isolation = Serializable$
                $\Rightarrow transaction[proposal[t][i].dependency.index].status \in \{Validated, Committed, Applied, A$
          $\land transaction' = [transaction \text{ EXCEPT } ![i].phase = Validate,$
                                                $![i].state \ \ = InProgress]$
          $\land \text{UNCHANGED } \langle proposal \rangle$
        If the transaction failed initialization, proceed to the *Abort* phase
        to ensure indexes are still updated for the target configurations.
        $\lor \land transaction[i].state = Failed$
          $\land transaction' = [transaction \text{ EXCEPT } ![i].phase = Abort,$
                                                $![i].state \ \ = InProgress]$
          $\land \text{UNCHANGED } \langle proposal \rangle$
$\lor \land transaction[i].phase = Validate$
  $\land \lor \land transaction[i].state = InProgress$
              Move the transaction's proposals to the Validating state
          $\land \lor \land \exists t \in transaction[i].targets :$
                      $\land proposal[t][i].phase \neq Validate$
                      $\land proposal' = [proposal \text{ EXCEPT } ![t] =$
                                 $[proposal[t] \text{ EXCEPT } ![i].phase = Validate,$
                                                          $![i].state \ \ = InProgress]]$
              $\land \text{UNCHANGED } \langle transaction \rangle$
              If all proposals have been *Complete*, mark the transaction *Complete*.
            $\lor \land \forall t \in transaction[i].targets :$
                      $\land proposal[t][i].phase \ \ = Validate$
                      $\land proposal[t][i].state = Complete$
              $\land transaction' = [transaction \text{ EXCEPT } ![i].state \ \ = Complete,$
                                                     $![i].status = Validated]$
              $\land \text{UNCHANGED } \langle proposal \rangle$
              If any proposal has been *Failed*, mark the transaction *Failed*.
            $\lor \land \exists t \in transaction[i].targets :$
                      $\land proposal[t][i].phase \ \ = Validate$
                      $\land proposal[t][i].state = Failed$
              $\land transaction' = [transaction \text{ EXCEPT } ![i].state = Failed]$
              $\land \text{UNCHANGED } \langle proposal \rangle$
        Once the transaction has been *Validated*, proceed to the *Commit* phase.
        If any of the transaction's proposals depend on a *Serializable* transaction,
        verify the dependency has been *Committed* to preserve serializability before
        moving the transaction to the *Commit* phase.
        $\lor \land transaction[i].state = Complete$
```

```
          $\land \forall t \in transaction[i].targets :$
              $\land proposal[t][i].dependency.index \in \textsc{domain}\ transaction$
              $\land transaction[proposal[t][i].dependency.index].isolation = Serializable$
              $\Rightarrow transaction[proposal[t][i].dependency.index].status \in \{Committed, Applied, Aborted\}$
          $\land transaction' = [transaction\ \textsc{except}\ ![i].phase = Commit,$
                                                   $![i].state\ = InProgress]$
          $\land \textsc{unchanged}\ \langle proposal \rangle$
        If the transaction failed validation, proceed to the Abort phase
        to ensure indexes are still updated for the target configurations.
        $\lor \land transaction[i].state = Failed$
          $\land transaction' = [transaction\ \textsc{except}\ ![i].phase = Abort,$
                                                   $![i].state\ = InProgress]$
          $\land \textsc{unchanged}\ \langle proposal \rangle$
$\lor \land transaction[i].phase = Commit$
   $\land \lor \land transaction[i].state = InProgress$
              Move the transaction's proposals to the Committing state
          $\land \lor \land \exists t \in transaction[i].targets :$
                      $\land proposal[t][i].phase \neq Commit$
                      $\land proposal' = [proposal\ \textsc{except}\ ![t] =$
                                  $[proposal[t]\ \textsc{except}\ ![i].phase = Commit,$
                                                          $![i].state\ = InProgress]]$
                 $\land \textsc{unchanged}\ \langle transaction \rangle$
              If all proposals have been Complete, mark the transaction Complete.
              $\lor \land \forall t \in transaction[i].targets :$
                      $\land proposal[t][i].phase\ = Commit$
                      $\land proposal[t][i].state = Complete$
                 $\land transaction' = [transaction\ \textsc{except}\ ![i].state\ = Complete,$
                                                          $![i].status = Committed]$
                 $\land \textsc{unchanged}\ \langle proposal \rangle$
        Once the transaction has been Committed, proceed to the Apply phase.
        If any of the transaction's proposals depend on a Serializable transaction,
        verify the dependency has been Applied to preserve serializability before
        moving the transaction to the Apply phase.
        $\lor \land transaction[i].state = Complete$
          $\land \forall t \in transaction[i].targets :$
              $\land proposal[t][i].dependency.index \in \textsc{domain}\ transaction$
              $\land transaction[proposal[t][i].dependency.index].isolation = Serializable$
              $\Rightarrow transaction[proposal[t][i].dependency.index].status \in \{Applied, Aborted\}$
          $\land transaction' = [transaction\ \textsc{except}\ ![i].phase = Apply,$
                                                   $![i].state\ = InProgress]$
          $\land \textsc{unchanged}\ \langle proposal \rangle$
$\lor \land transaction[i].phase = Apply$
   $\land transaction[i].state\ = InProgress$
        Move the transaction's proposals to the Applying state
   $\land \lor \land \exists t \in transaction[i].targets :$
```

```
                    $\land\ proposal[t][i].phase \neq Apply$
                    $\land\ proposal' = [proposal\ \text{EXCEPT}\ ![t] =$
                                        $[proposal[t]\ \text{EXCEPT}\ ![i].phase = Apply,$
                                                                   $![i].state\ \ = InProgress]]$
                $\land\ \text{UNCHANGED}\ \langle transaction \rangle$
            If all proposals have been Complete, mark the transaction Complete.
        $\lor\ \land\ \forall\, t \in transaction[i].targets :$
                    $\land\ proposal[t][i].phase\ \ = Apply$
                    $\land\ proposal[t][i].state = Complete$
            $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state\ \ = Complete,$
                                                           $![i].status = Applied]$
            $\land\ \text{UNCHANGED}\ \langle proposal \rangle$
            If any proposal has been Failed, mark the transaction Failed.
        $\lor\ \land\ \exists\, t \in transaction[i].targets :$
                    $\land\ proposal[t][i].phase\ \ = Apply$
                    $\land\ proposal[t][i].state = Failed$
            $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state = Failed]$
            $\land\ \text{UNCHANGED}\ \langle proposal \rangle$
    The Aborting state is used to clean up transactions that have failed during
    the Initializing or Validating phases.
    $\lor\ \land\ transaction[i].phase = Abort$
        $\land\ transaction[i].state\ \ = InProgress$
            Move the transaction's proposals to the Aborting state
        $\land\ \lor\ \land\ \exists\, t \in transaction[i].targets :$
                    $\land\ proposal[t][i].phase \neq Abort$
                    $\land\ proposal' = [proposal\ \text{EXCEPT}\ ![t] =$
                                        $[proposal[t]\ \text{EXCEPT}\ ![i].phase = Abort,$
                                                                   $![i].state\ \ = InProgress]]$
                $\land\ \text{UNCHANGED}\ \langle transaction \rangle$
            If all proposals have been Complete, mark the transaction Complete.
            $\lor\ \land\ \forall\, t \in transaction[i].targets :$
                    $\land\ proposal[t][i].phase\ \ = Abort$
                    $\land\ proposal[t][i].state = Complete$
                $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state\ \ = Complete,$
                                                               $![i].status = Aborted]$
                $\land\ \text{UNCHANGED}\ \langle proposal \rangle$
  $\land\ \text{UNCHANGED}\ \langle configuration, mastership, target \rangle$
```