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

INSTANCE FiniteSets

INSTANCE Sequences

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

 An empty constant
CONSTANT Nil

 Transaction type constants
CONSTANTS
   Change,
   Rollback

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

Phase ≜
   {Initialize,
    Validate,
    Commit,
    Apply}

 Status constants
CONSTANTS
   InProgress,
   Complete,
   Failed

State ≜
   {InProgress,
    Complete,
    Failed}

 State constants
CONSTANTS
   Pending,
   Validated,
   Committed,
   Applied,
   Aborted

Status ≜
   {Pending,
    Validated,
    Committed,
    Applied,
    Aborted}

CONSTANTS
   Valid,
   Invalid

CONSTANTS
   Success,
   Failure

 The set of all nodes
CONSTANT Node

Empty ≜ [p ∈ {} ↦ [value ↦ Nil, delete ↦ FALSE]]
```

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

Test ≜ INSTANCE Test WITH
    File        ← "Transaction.log",
    CurrState ← [
        transactions  ↦ transaction,
        proposals     ↦ proposal,
        configuration ↦ configuration,
        mastership    ↦ mastership,
        $target \mapsto target],$
    $SuccState \leftarrow [$
        $transactions \mapsto transaction',$
        $proposals \mapsto proposal',$
        $configuration \mapsto configuration',$
        $mastership \mapsto mastership',$
        $target \mapsto target']$
```

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

```
$RequestChange(p, v) \triangleq$
    $\land transaction' = Append(transaction, [type \mapsto Change,$
                                              $change \mapsto (p :> [index \mapsto Len(transaction) + 1, value \mapsto v]),$
                                              $phase \mapsto Initialize,$
                                              $state \mapsto InProgress])$
    $\land \text{UNCHANGED } \langle proposal, configuration, mastership, target \rangle$
```

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

```
$RequestRollback(i) \triangleq$
    $\land transaction' = Append(transaction, [type \mapsto Rollback,$
                                              $rollback \mapsto i,$
                                              $phase \mapsto Initialize,$
                                              $state \mapsto InProgress])$
    $\land \text{UNCHANGED } \langle proposal, configuration, mastership, target \rangle$
```

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

```
$ReconcileTransaction(n, i) \triangleq$
    $\land i \in \text{DOMAIN } transaction$
        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.
          $\land \neg\exists j \in \textsc{domain}\ transaction :$
                $\land j < i$
                $\land transaction[j].phase = Initialize$
                $\land transaction[j].state = InProgress$
            If the proposal does not exist in the queue, create it.
          $\land \lor \land i \notin \textsc{domain}\ proposal$
                  Append a change proposal.
                $\land \lor \land transaction[i].type = Change$
                        $\land proposal' = proposal\ @@\ (i :> [$
                                  $type \mapsto Change,$
                                  $change \mapsto [$
                                      $index \mapsto i,$
                                      $values \mapsto transaction[i].change],$
                                  $rollback \mapsto [$
                                      $index \mapsto 0,$
                                      $values \mapsto Empty],$
                                  $phase \mapsto Initialize,$
                                  $state \mapsto InProgress])$
                        $\land \textsc{unchanged}\ \langle transaction \rangle$
                  Append a rollback proposal.
                  $\lor \land transaction[i].type = Rollback$
                          If the rollback index is a valid Change transaction,
                          initialize the proposal.
                        $\land \lor \land transaction[i].rollback \in \textsc{domain}\ transaction$
                              $\land transaction[transaction[i].rollback].type = Change$
                              $\land proposal' = proposal\ @@\ (i :> [$
                                        $type \mapsto Rollback,$
                                        $change \mapsto [$
                                            $index \mapsto 0,$
                                            $values \mapsto Empty],$
                                        $rollback \mapsto [$
                                            $index \mapsto transaction[i].rollback,$
                                            $values \mapsto Empty],$
                                        $phase \mapsto Initialize,$
                                        $state \mapsto InProgress])$
                              $\land \textsc{unchanged}\ \langle transaction \rangle$
                          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 proposal has been created, check for completion or failures.
          $\lor\ \land\ i \in \textsc{domain}\ proposal$
              If the proposal has been Complete, mark the transaction Complete.
            $\land\ \lor\ \land\ proposal[i].phase = Initialize$
                    $\land\ proposal[i].state\ \ = Complete$
                    $\land\ transaction' = [transaction\ \textsc{except}\ ![i].state = Complete]$
                    $\land\ \textsc{unchanged}\ \langle proposal \rangle$
              If the proposal has been Failed, mark the transaction Failed.
                $\lor\ \land\ proposal[i].phase = Initialize$
                    $\land\ proposal[i].state\ \ = Failed$
                    $\land\ transaction' = [transaction\ \textsc{except}\ ![i].state = Failed]$
                    $\land\ \textsc{unchanged}\ \langle proposal \rangle$
      Once the transaction has been Initialized, move it to the validate phase.
      $\lor\ \land\ transaction[i].state = Complete$
          $\land\ transaction' = [transaction\ \textsc{except}\ ![i].phase = Validate,$
                                                           $![i].state\ \ = InProgress]$
          $\land\ \textsc{unchanged}\ \langle proposal \rangle$
  $\lor\ \land\ transaction[i].phase = Validate$
      $\land\ \lor\ \land\ transaction[i].state = InProgress$
          Move the transaction's proposals to the Validating state
          $\land\ \lor\ \land\ proposal[i].phase \neq Validate$
                  $\land\ proposal' = [proposal\ \textsc{except}\ ![i].phase = Validate,$
                                                         $![i].state = InProgress]$
                  $\land\ \textsc{unchanged}\ \langle transaction \rangle$
          If the proposals is Complete, mark the transaction Complete.
              $\lor\ \land\ proposal[i].phase = Validate$
                  $\land\ proposal[i].state\ \ = Complete$
                  $\land\ transaction' = [transaction\ \textsc{except}\ ![i].state = Complete]$
                  $\land\ \textsc{unchanged}\ \langle proposal \rangle$
          If the proposal has been Failed, mark the transaction Failed.
              $\lor\ \land\ proposal[i].phase = Validate$
                  $\land\ proposal[i].state\ \ = Failed$
                  $\land\ transaction' = [transaction\ \textsc{except}\ ![i].state = Failed]$
                  $\land\ \textsc{unchanged}\ \langle proposal \rangle$
      Once the transaction has been Validated, move it to the commit phase.
      $\lor\ \land\ transaction[i].state = Complete$
          $\land\ transaction' = [transaction\ \textsc{except}\ ![i].phase = Commit,$
                                                           $![i].state\ \ = InProgress]$
          $\land\ \textsc{unchanged}\ \langle proposal \rangle$
  $\lor\ \land\ transaction[i].phase = Commit$
      $\land\ \lor\ \land\ transaction[i].state = InProgress$
```

```
                Move the transaction's proposals to the Committing state
             $\land$ $\lor$ $\land$ $proposal[i].phase \neq Commit$
                   $\land$ $proposal' = [proposal$ EXCEPT $![i].phase = Commit,$
                                                        $![i].state\ \ = InProgress]$
                   $\land$ UNCHANGED $\langle transaction \rangle$
                If all proposals have been *Complete*, mark the transaction *Complete*.
                $\lor$ $\land$ $proposal[i].phase = Commit$
                   $\land$ $proposal[i].state\ \ = Complete$
                   $\land$ $transaction' = [transaction$ EXCEPT $![i].state = Complete]$
                   $\land$ UNCHANGED $\langle proposal \rangle$
          Once the transaction has been *Committed*, proceed to the *Apply* phase.
          $\lor$ $\land$ $transaction[i].state = Complete$
             $\land$ $transaction' = [transaction$ EXCEPT $![i].phase = Apply,$
                                                         $![i].state\ \ = InProgress]$
             $\land$ UNCHANGED $\langle proposal \rangle$
      $\lor$ $\land$ $transaction[i].phase = Apply$
         $\land$ $transaction[i].state\ \ = InProgress$
          Move the transaction's proposals to the Applying state
         $\land$ $\lor$ $\land$ $proposal[i].phase \neq Apply$
             $\land$ $proposal' = [proposal$ EXCEPT $![i].phase = Apply,$
                                                  $![i].state\ \ = InProgress]$
             $\land$ UNCHANGED $\langle transaction \rangle$
          If the proposal has been *Complete*, mark the transaction *Complete*.
          $\lor$ $\land$ $proposal[i].phase = Apply$
             $\land$ $proposal[i].state\ \ = Complete$
             $\land$ $transaction' = [transaction$ EXCEPT $![i].state = Complete]$
             $\land$ UNCHANGED $\langle proposal \rangle$
          If the proposal has been *Failed*, mark the transaction *Failed*.
          $\lor$ $\land$ $proposal[i].phase\ \ = Apply$
             $\land$ $proposal[i].state = Failed$
             $\land$ $transaction' = [transaction$ EXCEPT $![i].state = Failed]$
             $\land$ UNCHANGED $\langle proposal \rangle$
    $\land$ UNCHANGED $\langle configuration, mastership, target \rangle$
```