```
──────────────── MODULE Transaction ────────────────
EXTENDS Proposal

INSTANCE Naturals

INSTANCE FiniteSets

LOCAL INSTANCE TLC
────────────────────────────────────────────────────

 Transaction type constants
CONSTANTS
   TransactionChange,
   TransactionRollback

 Transaction isolation constants
CONSTANTS
   ReadCommitted,
   Serializable

 Phase constants
CONSTANTS
   TransactionInitialize,
   TransactionValidate,
   TransactionAbort,
   TransactionCommit,
   TransactionApply

 Status constants
CONSTANTS
   TransactionInProgress,
   TransactionComplete,
   TransactionFailed

 State constants
CONSTANTS
   TransactionPending,
   TransactionValidated,
   TransactionCommitted,
   TransactionApplied,
   TransactionAborted

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

```
LOCAL $InitState$ $\triangleq$
    $[transactions \mapsto transaction,$
     $proposals \mapsto [t \in$ DOMAIN $proposal \mapsto proposal[t]]]$

LOCAL $NextState$ $\triangleq$
    $[transactions \mapsto transaction',$
     $proposals \mapsto proposal']$

LOCAL $Trace$ $\triangleq$ INSTANCE $Trace$ WITH
    $Module \leftarrow$ “Transaction”,
    $InitState \leftarrow InitState,$
    $NextState \leftarrow NextState$
```

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
$ReconcileTransaction(i)$ $\triangleq$
        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.
    $\land$ $\lor$ $\land$ $transaction[i].phase = TransactionInitialize$
          $\land$ $\lor$ $\land$ $transaction[i].state = TransactionInProgress$
                   All prior transaction must be initialized before proceeding
                   to initialize this transaction.
                $\land \neg\exists j \in$ DOMAIN $transaction$ :
                     $\land j < i$
                     $\land transaction[j].phase = TransactionInitialize$
                     $\land transaction[j].state = TransactionInProgress$
                  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.
                     $\land$ $\lor$ $\land$ $transaction[i].type = TransactionChange$
        $\land$ $transaction' = [transaction$ EXCEPT $![i].targets = $ DOMAIN $transaction[i].change]$
        $\land$ $proposal' = [t \in$ DOMAIN $proposal \mapsto$
               IF $t \in$ DOMAIN $transaction[i].change$ THEN
                   $proposal[t]$ @@ $(i :> [type \mapsto ProposalChange,$
                                          $change \mapsto$
                                            $[index \mapsto i,$
                                             $values \mapsto transaction[i].change[t]],$
                                          $rollback \mapsto$
                                            $[index \mapsto 0],$
                                          $dependency \mapsto [index \mapsto 0],$
                                          $phase \mapsto ProposalInitialize,$
                                          $state \mapsto ProposalInProgress])$
               ELSE
                   $proposal[t]]$
     If the transaction is a rollback, the targets affected are
     the targets of the change transaction being rolled back.
     $\lor$ $\land$ $transaction[i].type = TransactionRollback$
           If the rollback index is a valid Change transaction,
           initialize proposals for all of the Change targets.
        $\land$ $\lor$ $\land$ $transaction[i].rollback \in$ DOMAIN $transaction$
              $\land$ $transaction[transaction[i].rollback].type = TransactionChange$
              $\land$ $transaction' = [transaction$ EXCEPT $![i].targets =$
                                     DOMAIN $transaction[transaction[i].rollback].change]$
              $\land$ $proposal' = [t \in$ DOMAIN $proposal \mapsto$
                     IF $t \in$ DOMAIN $transaction[transaction[i].rollback].change$ THEN
                         $proposal[t]$ @@ $(i :> [type \mapsto ProposalRollback,$
                                                $change \mapsto$
                                                  $[index \mapsto 0],$
                                                $rollback \mapsto$
                                                  $[index \mapsto transaction[i].rollback],$
                                                $dependency \mapsto [index \mapsto 0],$
                                                $phase \mapsto ProposalInitialize,$
                                                $state \mapsto ProposalInProgress])$
                     ELSE
                         $proposal[t]]$
           If the rollback index is not a valid Change transaction
           fail the Rollback transaction.
           $\lor$ $\land$ $\lor$ $\land$ $transaction[i].rollback \in$ DOMAIN $transaction$
                    $\land$ $transaction[transaction[i].rollback].type = TransactionRollback$
                 $\lor$ $transaction[i].rollback \notin$ DOMAIN $transaction$
              $\land$ $transaction' = [transaction$ EXCEPT $![i].state = TransactionFailed]$
              $\land$ 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*.
        $\land \lor \land \forall t \in transaction[i].targets :$
                    $\land proposal[t][i].phase = ProposalInitialize$
                    $\land proposal[t][i].state\ = ProposalComplete$
                $\land transaction' = [transaction\ \text{EXCEPT}\ ![i].state = TransactionComplete]$
                $\land \text{UNCHANGED}\ \langle proposal \rangle$
            If any proposal has been *Failed*, mark the transaction *Failed*.
            $\lor \land \exists t \in transaction[i].targets :$
                    $\land proposal[t][i].phase = ProposalInitialize$
                    $\land proposal[t][i].state\ = ProposalFailed$
                $\land transaction' = [transaction\ \text{EXCEPT}\ ![i].state = TransactionFailed]$
                $\land \text{UNCHANGED}\ \langle proposal \rangle$
        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.
        $\lor \land transaction[i].state = TransactionComplete$
          $\land \forall t \in transaction[i].targets :$
              $\land proposal[t][i].dependency.index \in \text{DOMAIN}\ transaction$
              $\land transaction[proposal[t][i].dependency.index].isolation = Serializable$
              $\Rightarrow transaction[proposal[t][i].dependency.index].status$
                    $\in \{TransactionValidated, TransactionCommitted, TransactionApplied, TransactionA$
          $\land transaction' = [transaction\ \text{EXCEPT}\ ![i].phase = TransactionValidate,$
                                                   $![i].state\ = TransactionInProgress]$
          $\land \text{UNCHANGED}\ \langle proposal \rangle$
        If the transaction failed initialization, proceed to the *Abort* phase
        to ensure indexes are still updated for the target configurations.
        $\lor \land transaction[i].state = TransactionFailed$
          $\land transaction' = [transaction\ \text{EXCEPT}\ ![i].phase = TransactionAbort,$
                                                   $![i].state\ = TransactionInProgress]$
          $\land \text{UNCHANGED}\ \langle proposal \rangle$
    $\lor \land transaction[i].phase = TransactionValidate$
      $\land \lor \land transaction[i].state = TransactionInProgress$
            Move the transaction's proposals to the Validating state
          $\land \lor \land \exists t \in transaction[i].targets :$
                    $\land proposal[t][i].phase \neq ProposalValidate$
                    $\land proposal' = [proposal\ \text{EXCEPT}\ ![t] =$
                                $[proposal[t]\ \text{EXCEPT}\ ![i].phase = ProposalValidate,$
                                                  $![i].state\ = ProposalInProgress]]$
                $\land \text{UNCHANGED}\ \langle transaction \rangle$
            If all proposals have been *Complete*, mark the transaction *Complete*.
            $\lor \land \forall t \in transaction[i].targets :$
                    $\land proposal[t][i].phase = ProposalValidate$
                    $\land proposal[t][i].state\ = ProposalComplete$
                $\land transaction' = [transaction\ \text{EXCEPT}\ ![i].state\ = TransactionComplete,$
```

```
                                                          ![i].status = TransactionValidated]
              ∧ UNCHANGED ⟨proposal⟩
             If any proposal has been Failed, mark the transaction Failed.
             ∨ ∧ ∃ t ∈ transaction[i].targets :
                    ∧ proposal[t][i].phase = ProposalValidate
                    ∧ proposal[t][i].state  = ProposalFailed
               ∧ transaction′ = [transaction EXCEPT ![i].state = TransactionFailed]
               ∧ UNCHANGED ⟨proposal⟩
          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.
          ∨ ∧ transaction[i].state = TransactionComplete
            ∧ ∀ t ∈ transaction[i].targets :
                 ∧ proposal[t][i].dependency.index ∈ DOMAIN transaction
                 ∧ transaction[proposal[t][i].dependency.index].isolation = Serializable
                 ⇒ transaction[proposal[t][i].dependency.index].status
                        ∈ {TransactionCommitted, TransactionApplied, TransactionAborted}
            ∧ transaction′ = [transaction EXCEPT ![i].phase = TransactionCommit,
                                                 ![i].state  = TransactionInProgress]
            ∧ UNCHANGED ⟨proposal⟩
          If the transaction failed validation, proceed to the Abort phase
          to ensure indexes are still updated for the target configurations.
          ∨ ∧ transaction[i].state = TransactionFailed
            ∧ transaction′ = [transaction EXCEPT ![i].phase = TransactionAbort,
                                                 ![i].state  = TransactionInProgress]
            ∧ UNCHANGED ⟨proposal⟩
       ∨ ∧ transaction[i].phase = TransactionCommit
         ∧ ∨ ∧ transaction[i].state = TransactionInProgress
             Move the transaction's proposals to the Committing state
            ∧ ∨ ∧ ∃ t ∈ transaction[i].targets :
                    ∧ proposal[t][i].phase ≠ ProposalCommit
                    ∧ proposal′ = [proposal EXCEPT ![t] =
                                     [proposal[t] EXCEPT ![i].phase = ProposalCommit,
                                                         ![i].state  = ProposalInProgress]]
               ∧ UNCHANGED ⟨transaction⟩
             If all proposals have been Complete, mark the transaction Complete.
             ∨ ∧ ∀ t ∈ transaction[i].targets :
                    ∧ proposal[t][i].phase = ProposalCommit
                    ∧ proposal[t][i].state  = ProposalComplete
               ∧ transaction′ = [transaction EXCEPT ![i].state  = TransactionComplete,
                                                    ![i].status = TransactionCommitted]
               ∧ UNCHANGED ⟨proposal⟩
          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.
      $\lor \land transaction[i].state = TransactionComplete$
        $\land \forall t \in transaction[i].targets :$
            $\land proposal[t][i].dependency.index \in \textsc{domain}\ transaction$
            $\land transaction[proposal[t][i].dependency.index].isolation = Serializable$
            $\Rightarrow transaction[proposal[t][i].dependency.index].status$
                  $\in \{TransactionApplied, TransactionAborted\}$
        $\land transaction' = [transaction\ \textsc{except}\ ![i].phase = TransactionApply,$
                                                   $![i].state\ \ = TransactionInProgress]$
        $\land \textsc{unchanged}\ \langle proposal \rangle$
$\lor \land transaction[i].phase = TransactionApply$
  $\land transaction[i].state\ \ = TransactionInProgress$
        Move the transaction's proposals to the Applying state
  $\land \lor \land \exists t \in transaction[i].targets :$
            $\land proposal[t][i].phase \neq ProposalApply$
            $\land proposal' = [proposal\ \textsc{except}\ ![t] =$
                          $[proposal[t]\ \textsc{except}\ ![i].phase = ProposalApply,$
                                                $![i].state\ \ = ProposalInProgress]]$
        $\land \textsc{unchanged}\ \langle transaction \rangle$
        If all proposals have been *Complete*, mark the transaction *Complete*.
      $\lor \land \forall t \in transaction[i].targets :$
            $\land proposal[t][i].phase = ProposalApply$
            $\land proposal[t][i].state\ \ = ProposalComplete$
        $\land transaction' = [transaction\ \textsc{except}\ ![i].state\ \ = TransactionComplete,$
                                                   $![i].status = TransactionApplied]$
        $\land \textsc{unchanged}\ \langle proposal \rangle$
        If any proposal has been *Failed*, mark the transaction *Failed*.
      $\lor \land \exists t \in transaction[i].targets :$
            $\land proposal[t][i].phase = ProposalApply$
            $\land proposal[t][i].state\ \ = ProposalFailed$
        $\land transaction' = [transaction\ \textsc{except}\ ![i].state = TransactionFailed]$
        $\land \textsc{unchanged}\ \langle proposal \rangle$
  The Aborting state is used to clean up transactions that have failed during
  the Initializing or Validating phases.
$\lor \land transaction[i].phase = TransactionAbort$
  $\land transaction[i].state\ \ = TransactionInProgress$
        Move the transaction's proposals to the Aborting state
  $\land \lor \land \exists t \in transaction[i].targets :$
            $\land proposal[t][i].phase \neq ProposalAbort$
            $\land proposal' = [proposal\ \textsc{except}\ ![t] =$
                          $[proposal[t]\ \textsc{except}\ ![i].phase = ProposalAbort,$
                                                $![i].state\ \ = ProposalInProgress]]$
        $\land \textsc{unchanged}\ \langle transaction \rangle$
        If all proposals have been *Complete*, mark the transaction *Complete*.
```

```
$\lor\ \land \forall\, t \in transaction[i].targets :$
        $\land\ proposal[t][i].phase\ \ = ProposalAbort$
        $\land\ proposal[t][i].state = ProposalComplete$
   $\land\ transaction' = [transaction$ EXCEPT $![i].state\ \ = TransactionComplete,$
                                               $![i].status = TransactionAborted]$
   $\land$ UNCHANGED $\langle proposal \rangle$
```

Formal specification, constraints, and theorems.

```
$InitTransaction \triangleq$
    $\land\ transaction = [i \in \{\} \mapsto$
                              $[type\ \ \ \mapsto TransactionChange,$
                               $phase \mapsto TransactionInitialize,$
                               $state\ \ \mapsto TransactionInProgress,$
                               $status \mapsto TransactionPending]]$
    $\land\ Trace!Init$

$NextTransaction \triangleq$
    $\lor\ \exists\, i \in$ DOMAIN $transaction :$
        $Trace!Step($"Reconcile"$, ReconcileTransaction(i), [index \mapsto i])$
```

* Modification History
* Last modified Sun *Feb* 20 09:00:05 *PST* 2022 by *jordanhalterman*
* Created Sun *Feb* 20 02:20:45 *PST* 2022 by *jordanhalterman*