MODULE *Transaction*

INSTANCE *Naturals*

INSTANCE *FiniteSets*

LOCAL INSTANCE *TLC*

Transaction type constants

CONSTANTS
*Change*,
*Rollback*

Transaction isolation constants

CONSTANTS
*ReadCommitted*,
*Serializable*

Phase constants

CONSTANTS
*Initialize*,
*Validate*,
*Abort*,
*Commit*,
*Apply*

Status constants

CONSTANTS
*InProgress*,
*Complete*,
*Failed*

State constants

CONSTANTS
*Pending*,
*Validated*,
*Committed*,
*Applied*,
*Aborted*

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*

LOCAL $InitState \triangleq$
$[transactions \mapsto transaction,$
$\ proposals \quad \mapsto [t \in \text{DOMAIN } proposal \mapsto proposal[t]]]$

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

LOCAL $Trace \triangleq$ INSTANCE $Trace$ WITH
$Module \quad \leftarrow$ "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
$Reconcile(i) \triangleq$

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.
  ∧ ∨ ∧ transaction[i].type = Change
      ∧ transaction' = [transaction EXCEPT ![i].targets = DOMAIN transaction[i].change]
      ∧ proposal' = [t ∈ DOMAIN proposal ↦
            IF t ∈ DOMAIN transaction[i].change THEN
               proposal[t] @@ (i :> [type       ↦ Change,
                                     change     ↦
                                       [index   ↦ i,
                                        values  ↦ transaction[i].change[t]],
                                     rollback   ↦
                                       [index   ↦ 0],
                                     dependency ↦ [index ↦ 0],
                                     phase      ↦ Initialize,
                                     state      ↦ InProgress])
            ELSE
               proposal[t]]
                If the transaction is a rollback, the targets affected are
                the targets of the change transaction being rolled back.
      ∨ ∧ transaction[i].type = Rollback
                If the rollback index is a valid Change transaction,
                initialize proposals for all of the Change targets.
        ∧ ∨ ∧ transaction[i].rollback ∈ DOMAIN transaction
            ∧ transaction[transaction[i].rollback].type = Change
            ∧ transaction' = [transaction EXCEPT ![i].targets =
                                 DOMAIN transaction[transaction[i].rollback].change]
            ∧ proposal' = [t ∈ DOMAIN proposal ↦
                  IF t ∈ DOMAIN transaction[transaction[i].rollback].change THEN
                     proposal[t] @@ (i :> [type       ↦ Rollback,
                                           change     ↦
                                             [index   ↦ 0],
                                           rollback   ↦
                                             [index   ↦ transaction[i].rollback],
                                           dependency ↦ [index ↦ 0],
                                           phase      ↦ Initialize,
                                           state      ↦ InProgress])
                  ELSE
                     proposal[t]]
                If the rollback index is not a valid Change transaction
                fail the Rollback transaction.
          ∨ ∧ ∨ ∧ transaction[i].rollback ∈ DOMAIN transaction
                ∧ transaction[transaction[i].rollback].type = Rollback
              ∨ transaction[i].rollback ∉ DOMAIN transaction
            ∧ transaction' = [transaction EXCEPT ![i].state = Failed]
            ∧ UNCHANGED ⟨proposal⟩
If the transaction's proposals have been initialized, check proposals
```

```
                for completion or failures.
            ∨ ∧ transaction[i].targets ≠ {}
                    If all proposals have been Complete, mark the transaction Complete.
                ∧ ∨ ∧ ∀ t ∈ transaction[i].targets :
                          ∧ proposal[t][i].phase = Initialize
                          ∧ proposal[t][i].state  = Complete
                    ∧ transaction' = [transaction EXCEPT ![i].state = Complete]
                    ∧ UNCHANGED ⟨proposal⟩
                    If any proposal has been Failed, mark the transaction Failed.
                  ∨ ∧ ∃ t ∈ transaction[i].targets :
                          ∧ proposal[t][i].phase  = Initialize
                          ∧ proposal[t][i].state = Failed
                    ∧ transaction' = [transaction EXCEPT ![i].state = Failed]
                    ∧ UNCHANGED ⟨proposal⟩
            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.
          ∨ ∧ transaction[i].state = Complete
            ∧ ∀ 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 ∈ {Validated, Committed, Applied, A
            ∧ transaction' = [transaction EXCEPT ![i].phase = Validate,
                                                 ![i].state  = InProgress]
            ∧ UNCHANGED ⟨proposal⟩
            If the transaction failed initialization, proceed to the Abort phase
            to ensure indexes are still updated for the target configurations.
          ∨ ∧ transaction[i].state = Failed
            ∧ transaction' = [transaction EXCEPT ![i].phase = Abort,
                                                 ![i].state  = InProgress]
            ∧ UNCHANGED ⟨proposal⟩
      ∨ ∧ transaction[i].phase = Validate
        ∧ ∨ ∧ transaction[i].state = InProgress
                Move the transaction's proposals to the Validating state
            ∧ ∨ ∧ ∃ t ∈ transaction[i].targets :
                      ∧ proposal[t][i].phase ≠ Validate
                      ∧ proposal' = [proposal EXCEPT ![t] =
                                        [proposal[t] EXCEPT ![i].phase = Validate,
                                                            ![i].state  = InProgress]]
              ∧ UNCHANGED ⟨transaction⟩
              If all proposals have been Complete, mark the transaction Complete.
            ∨ ∧ ∀ t ∈ transaction[i].targets :
                      ∧ proposal[t][i].phase  = Validate
                      ∧ proposal[t][i].state = Complete
```

$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state\ = Complete,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![i].status = Validated]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

If any proposal has been *Failed*, mark the transaction *Failed*.

$\lor\ \land \exists\, t \in transaction[i].targets :$
$\qquad\land\ proposal[t][i].phase\ = Validate$
$\qquad\land\ proposal[t][i].state = Failed$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state = Failed]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

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.

$\lor\ \land transaction[i].state = Complete$
$\land \forall\, t \in transaction[i].targets :$
$\qquad\land\ proposal[t][i].dependency.index \in \text{DOMAIN}\ transaction$
$\qquad\land\ transaction[proposal[t][i].dependency.index].isolation = Serializable$
$\qquad\Rightarrow transaction[proposal[t][i].dependency.index].status \in \{Committed,\ Applied,\ Aborted\}$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].phase = Commit,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![i].state\ = InProgress]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

If the transaction failed validation, proceed to the *Abort* phase
to ensure indexes are still updated for the target configurations.

$\lor\ \land transaction[i].state = Failed$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].phase = Abort,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![i].state\ = InProgress]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

$\lor\ \land transaction[i].phase = Commit$
$\land\ \lor\ \land transaction[i].state = InProgress$

Move the transaction's proposals to the Committing state

$\land\ \lor\ \land \exists\, t \in transaction[i].targets :$
$\qquad\land\ proposal[t][i].phase \neq Commit$
$\qquad\land\ proposal' = [proposal\ \text{EXCEPT}\ ![t] =$
$\qquad\qquad\qquad [proposal[t]\ \text{EXCEPT}\ ![i].phase = Commit,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![i].state\ = InProgress]]$
$\qquad\land\ \text{UNCHANGED}\ \langle transaction \rangle$

If all proposals have been *Complete*, mark the transaction *Complete*.

$\qquad\lor\ \land \forall\, t \in transaction[i].targets :$
$\qquad\qquad\land\ proposal[t][i].phase\ = Commit$
$\qquad\qquad\land\ proposal[t][i].state = Complete$
$\qquad\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state\ = Complete,$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![i].status = Committed]$
$\qquad\land\ \text{UNCHANGED}\ \langle proposal \rangle$

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.
      ∨ ∧ transaction[i].state = Complete
        ∧ ∀ 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 ∈ {Applied, Aborted}
        ∧ transaction' = [transaction EXCEPT ![i].phase = Apply,
                                             ![i].state  = InProgress]
        ∧ UNCHANGED ⟨proposal⟩
  ∨ ∧ transaction[i].phase = Apply
    ∧ transaction[i].state  = InProgress
        Move the transaction's proposals to the Applying state
    ∧ ∨ ∧ ∃ t ∈ transaction[i].targets :
            ∧ proposal[t][i].phase ≠ Apply
            ∧ proposal' = [proposal EXCEPT ![t] =
                                [proposal[t] EXCEPT ![i].phase = Apply,
                                                    ![i].state  = InProgress]]
        ∧ UNCHANGED ⟨transaction⟩
        If all proposals have been Complete, mark the transaction Complete.
      ∨ ∧ ∀ t ∈ transaction[i].targets :
            ∧ proposal[t][i].phase  = Apply
            ∧ proposal[t][i].state = Complete
        ∧ transaction' = [transaction EXCEPT ![i].state  = Complete,
                                             ![i].status = Applied]
        ∧ UNCHANGED ⟨proposal⟩
        If any proposal has been Failed, mark the transaction Failed.
      ∨ ∧ ∃ t ∈ transaction[i].targets :
            ∧ proposal[t][i].phase  = Apply
            ∧ proposal[t][i].state = Failed
        ∧ transaction' = [transaction EXCEPT ![i].state = Failed]
        ∧ UNCHANGED ⟨proposal⟩
  The Aborting state is used to clean up transactions that have failed during
  the Initializing or Validating phases.
  ∨ ∧ transaction[i].phase = Abort
    ∧ transaction[i].state  = InProgress
        Move the transaction's proposals to the Aborting state
    ∧ ∨ ∧ ∃ t ∈ transaction[i].targets :
            ∧ proposal[t][i].phase ≠ Abort
            ∧ proposal' = [proposal EXCEPT ![t] =
                                [proposal[t] EXCEPT ![i].phase = Abort,
                                                    ![i].state  = InProgress]]
        ∧ UNCHANGED ⟨transaction⟩
        If all proposals have been Complete, mark the transaction Complete.
      ∨ ∧ ∀ t ∈ transaction[i].targets :
```

$$
\begin{aligned}
&\land\ proposal[t][i].phase\ \ = Abort \\
&\land\ proposal[t][i].state = Complete \\
&\land\ transaction' = [transaction \text{ EXCEPT } ![i].state\ \ = Complete, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![i].status = Aborted] \\
&\land \text{ UNCHANGED } \langle proposal \rangle
\end{aligned}
$$

---

Formal specification, constraints, and theorems.

$$
\begin{aligned}
&Init\ \triangleq \\
&\quad \land\ transaction = [i \in \{\} \mapsto \\
&\qquad\qquad\qquad\qquad [type\ \ \mapsto Change, \\
&\qquad\qquad\qquad\qquad\ phase\ \mapsto Initialize, \\
&\qquad\qquad\qquad\qquad\ state\ \ \mapsto InProgress, \\
&\qquad\qquad\qquad\qquad\ status \mapsto Pending]] \\
&\quad \land\ Trace!Init \\
&\\
&Next\ \triangleq \\
&\quad \lor \exists\, i \in \text{DOMAIN } transaction : \\
&\qquad\ Trace!Step(\text{“Reconcile”},\ Reconcile(i),\ [index \mapsto i])
\end{aligned}
$$

---

* Modification History
* Last modified Sun *Feb* 20 08:19:22 *PST* 2022 by *jordanhalterman*
* Created Sun *Feb* 20 02:20:45 *PST* 2022 by *jordanhalterman*