MODULE *TwoPhaseLocking*

EXTENDS *FiniteSets*, *Naturals*, *Sequences*, *TLC*

CONSTANT *Proc*, *Object*

VARIABLE
  transact,
  history,
  state,
  store,
  READ, read lock
  WRITE write lock

$vars \triangleq \langle$
  transact,
  history,
  state,
  store,
  READ,
  WRITE
$\rangle$

Transaction is a set of all possible transactions

$Transaction \triangleq$
  LET $Op \triangleq [f : \{\text{"Read"}, \text{"Write"}\}, obj : Object]$
      $seq(S) \triangleq \text{UNION } \{[1 \,..\, n \to S] : n \in Nat\}$
  IN $\{Append(op, [f \mapsto \text{"Commit"}]) : op \in seq(Op)\}$

$Init \triangleq$
  $\land \exists\, tx \in [Proc \to Transaction] : transact = tx$
  $\land history = \langle\rangle$
  $\land state = [proc \in Proc \mapsto \text{"Init"}]$
  $\land store = [obj \in Object \mapsto 0]$
  $\land READ = [obj \in Object \mapsto \{\}]$
  $\land WRITE = [obj \in Object \mapsto \{\}]$

$updateHistory(self, hd, tl, val) \triangleq$
  $\land history' = Append(history, [proc \mapsto self, op \mapsto hd, val \mapsto val])$
  $\land transact' = [transact \text{ EXCEPT } ![self] = tl]$

$ReadLongDurationLock(self, hd, tl) \triangleq$
  $\land state[self] \in \{\text{"Init"}, \text{"Running"}\}$
  $\land hd.f = \text{"Read"}$
  $\land WRITE[hd.obj] \in \{\{\}, \{self\}\}$
  $\land READ' = [READ \text{ EXCEPT } ![hd.obj] = READ[hd.obj] \cup \{self\}]$
  $\land updateHistory(self, hd, tl, store[hd.obj])$
  $\land \text{IF } state[self] = \text{"Init"}$

```
      THEN  ∧ state' = [state EXCEPT ![self] = "Running"]
            ∧ UNCHANGED ⟨store, WRITE⟩
      ELSE  UNCHANGED ⟨state, store, WRITE⟩

ReadShortDurationLock(self, hd, tl) ≜
  ∧ state[self] ∈ {"Init", "Running"}
  ∧ hd.f = "Read"
  ∧ WRITE[hd.obj] ∈ {{}, {self}}
  ∧ updateHistory(self, hd, tl, store[hd.obj])
  ∧ IF state[self]    = "Init"
      THEN  ∧ state' = [state EXCEPT ![self] = "Running"]
            ∧ UNCHANGED ⟨store, READ, WRITE⟩
      ELSE  UNCHANGED ⟨state, store, READ, WRITE⟩

Read(self, hd, tl) ≜ ReadLongDurationLock(self, hd, tl)

Write(self, hd, tl) ≜
  ∧   state[self] ∈ {"Init", "Running"}
  ∧   hd.f = "Write"
  ∧   WRITE[hd.obj] ∈ {{}, {self}}
  ∧   WRITE' = [WRITE EXCEPT ![hd.obj] = WRITE[hd.obj] ∪ {self}]
  ∧   READ[hd.obj] ∈ SUBSET WRITE'[hd.obj]
  ∧   store' = [store EXCEPT ![hd.obj] = store[hd.obj] + 1]
  ∧   updateHistory(self, hd, tl, store[hd.obj] + 1)
  ∧   IF state[self]    = "Init"
        THEN  ∧ state' = [state EXCEPT ![self] = "Running"]
              ∧ UNCHANGED ⟨READ⟩
        ELSE  UNCHANGED ⟨state, READ⟩

Commit(self, hd, tl) ≜
  ∧ state[self] ∈ {"Init", "Running"}
  ∧ hd.f = "Commit"
  ∧ updateHistory(self, hd, tl, 0)
  ∧ state' = [state EXCEPT ![self] = "Commit"]
  ∧ READ' = [obj ∈ Object ↦ READ[obj] \ {self}]
  ∧ WRITE' = [obj ∈ Object ↦ WRITE[obj] \ {self}]
  ∧ UNCHANGED ⟨store⟩

Next ≜ ∃ self ∈ Proc
  : ∧ transact[self] ≠ ⟨⟩
    ∧ LET hd ≜ Head(transact[self])
          tl ≜ Tail(transact[self])
      IN    ∨ Read(self, hd, tl)
            ∨ Write(self, hd, tl)
            ∨ Commit(self, hd, tl)

Spec ≜ Init ∧ □[Next]_vars ∧ WF_vars(Next)
```

```
Invariants ≜
  ∧ ∀ proc ∈ Proc
      : state[proc] ∈ {"Init", "Running", "Commit"}
  ∧ ∀ obj ∈ Object
      : Cardinality(WRITE[obj]) ∈ {0, 1}
  ∧ ∀ obj ∈ Object
      : Cardinality(WRITE[obj]) ≠ 0 ⇒  READ[obj] ∈ SUBSET WRITE[obj]
```

Serializable tests if a history is serializable

```
RECURSIVE consistent(_, _)
consistent(s, hist) ≜
  IF hist = ⟨⟩
   THEN TRUE
   ELSE LET hd ≜ Head(hist)
          IN  CASE hd.op.f = "Read"
                   → s[hd.op.obj] = hd.val ∧ consistent(s, Tail(hist))
                □  hd.op.f = "Write"
                   → consistent([s EXCEPT ![hd.op.obj] = hd.val], Tail(hist))
                □  OTHER
                   → consistent(s, Tail(hist))

Serializable ≜
  LET Tx ≜ {SelectSeq(history, LAMBDA x : x.proc = proc) : proc ∈ Proc}
      perms ≜ {f ∈ [1 .. Cardinality(Proc) → Tx]
                 : ∀ tx ∈ Tx
                   : ∃ proc ∈ 1 .. Cardinality(Proc) : f[proc] = tx}
  IN  LET RECURSIVE concat(_, _, _, _)
          concat(f, n, size, acc) ≜
            IF n > size THEN acc ELSE  concat(f, n + 1, size, acc ∘ f[n])
      IN  ∃ perm ∈ perms
          : consistent([obj ∈ Object ↦ 0],
                       concat(perm, 1, Cardinality(Proc), ⟨⟩))
    ∧ PrintT(⟨history, concat(perm, 1, Cardinality(Proc), ⟨⟩)⟩)
```

Properties assert that if all transactions successfully commit then the history is serializable

```
Properties ≜
  □((∀ proc ∈ Proc : state[proc] = "Commit") ⇒ Serializable)
```

Deadlock detects a deadlock

```
Deadlock ≜ ◇□(∀ proc ∈ Proc : state[proc] = "Commit")

THEOREM Spec ⇒ □Invariants ∧ Properties
```

```
\* Modification History
\* Last modified Sat Feb 17 13:01:22 JST 2018 by takayuki
\* Created Sat Feb 17 10:34:44 JST 2018 by takayuki
```