```
MODULE ConcurrentTransactions [
   V, % Value
   S, % State of database
   T  % Transaction ID
] EXPORT Begin, Do, Commit =

CONST s0:S := init()    % initial state

TYPE
     A = S -> [v,s]     % Action
     E = [a: A, v: V]   % Event
     H = SEQ E          % History
     Y = T -> H         % histories of transactions

     TS = SET T         % Transaction Set
     XC = (T, T)-> Bool % eXternal Consistency
     TO = SEQ T         % Total Order on T's

VAR
    y := Y{}             % current transaction histories
    committed : TS{}     % committed transactions
    active    : TS{}     % active transactions (uncommited)
    xc := XC{* -> false} % current required XC
```

```
FUNC Valid(y,to) -> Bool = RET Apply(+ : (to * y),s0)

FUNC Apply(h: H, s: S) =
 IF  h={}  => RET True
 [*] VAR [a,v]   := h.head |
     VAR [v',s'] := a(s)   |
     IF ~(v'=v) => RET False
     [*] RET Apply(h.tail, s')
 FI

FUNC Consistent(to, xc) -> Bool =
  ALL t1, t2: T |
    xc.closure(t1,t2) ==> ( TO{t1,t2} <<= to)

FUNC Serializable(ts: TS, xc: XC, y: Y) -> Bool =
  RET (EXISTS to: TO |
        t.set=ts /\ Consistent(to, xc) /\ Valid(y,to))

FUNC Invariant(com: TS, act: TS, xc, y) -> Bool =
  Serializable(com, xc, y)
```

```
APROC Begin() -> T = <<
  VAR t: T | ~ t IN (active \/ committed) =>
    y(t) := {};
    active := active \/ {t};
    xc(t,t) := true;
    DO VAR t' :IN committed | ~xc.closure(t',t) =>
      xc(t',t):=true
    OD;
>>

APROC Do(t: T, a: A) -> V = <<
  VAR v: V,
      y' := t{t -> t(y) + {E{a,v}}} |
  Invariant(commited, active, xc, y') =>
    y := y';
    RET v;
>>

APROC Commit(t: T) = <<
  VAR committed' :TS := commited \/ {t},
      active'    :TS := active - {t} |
  Invariant(commited', active', xc, y) =>
    commited := commited'
    active   := active';
>>

% INVARIANT Invariant(commited, active, xc, y)
```

```
FUNC Invariant(com: TS, act: TS, xc0: XC, y0: Y) -> Bool =
VAR current := com + act |
  Serializable(com, xc0, y0) /\ CONSTRAINT

CONSTRAINT is one of AC, CC, EO, OD, OC1, OC2, NC

AC:  (ALL ts: TS | (com <= ts <= current) ==>
        Serializable(ts, xc0, y0))

CC:   Serializable(current, xc0, y0)

EO:  (ALL t :IN act | EXISTS ts | com <= ts <= current /\
        Serializable(ts + {t}, xc0, y0))

OD:  (ALL t :IN act | EXISTS ts | AtBegin(t) <= ts <= current /\
        Serializable(ts + {t}, xc0, y0))

OC1: (ALL t :IN act, h :IN Prefixes(y0(t)) | EXISTS to, h1, h2 |
       to.set = AtBegin(t) /\ Consistent(to, xc0) /\ Valid(y0,to)
    /\ IsInterleaving(h1, {t' | t' IN current-AtBegin(t)-{t} | y0(t')})
    /\ h2 <<= h1
    /\ h.last.a(Apply(+ : (to * y0) + h2 + h.reml, s0) = h.last.v))

OC2: (ALL t :IN act, h :IN Prefixes(y0(t)) | EXISTS to, h1, h2, h3 |
       to.set = AtBegin(t) /\ Consistent(to, xc0) /\ Valid(y0,to)
    /\ IsInterleaving(h1, {t' | t' IN current-AtBegin(t)-{t} | y0(t')})
    /\ h2 <<= h1
    /\ IsInterleaving(h3, {h2, h.reml})
    /\ h.last.a(Apply(+ : (to * y0) + h2 + h.reml, s0) = h.last.v))

NC: true


FUNC Prefixes(h: T) -> SET H = RET {h' | h' M= h /\ h' # {}}

FUNC AtBegin(t: T) -> TS = RET {t' | xc.closure(t',t)}

FUNC IsInterleaving(h: H, s: SET H) -> Bool =
  ... sequence h is interleaving of sequences from the set s ...
```

```
TYPE Lk    = String
     Lks   = SET Lk
     A     = S -> [v: V, s: S]

CONST
   protect    : A -> Lks
   conflict   : (Lk, Lk) -> Bool

% I1:  ALL a1, a2 | ~commute(a1,a2) ==>
         EXISTS l1 IN protect(a1), l2 IN protect(a2) |
           conflict(l1, l2)

FUNC commute(a1: A, a2: A) -> Bool =
  RET (ALL s0: S |
       a2(a1(s0).s).s = a1(a2(s0).s).s /\
       a1(s0).v = a1(a2(s0).s).v /\
       a2(s0).v = a2(a1(s0).s).v)

% I2:  ALL t :IN active, e :IN y(t) | protect(e.a) <= locks(t)

% I3:  ALL t1 :IN active, t2 :IN active | t1 # t2 ==>
%         ALL lk1 :IN locks(t1), lk2 :IN locks(t2) |
%           ~conflict(lk1,lk2)
```