```
──────────────────────── MODULE SSI ────────────────────────
Serializable Snapshot Isolation

Based on the algorithm described in the paper:

Serializable Isolation for Snapshot Databases, Michael J. Cahill, Uwe Röhm, Alan D. Fekete,
SIGMOD′08, June 2008.

EXTENDS MVCC

VARIABLES rds,    which transactions have performed reads on each object
          outc,   transactions that have an outbound anti-dependency
          inc     transactions that have an inbound anti-dependency

TypeOkS ≜  ∧ rds ∈ [Obj → SUBSET Tr]
           ∧ outc ⊆ Tr
           ∧ inc ⊆ Tr

InitS ≜  ∧ Init
         ∧ rds = [obj ∈ Obj ↦ {}]
         ∧ inc = {}
         ∧ outc = {}

BeginRdS(t, obj) ≜
   LET  isActiveWrite ≜ (∃ tw ∈ Tr \ {t} : ActiveWrite(tw, obj))
        tw ≜ CHOOSE tw ∈ Tr \ {t} : ActiveWrite(tw, obj)
        localWriteToObj ≜ ∃ ver ∈ db[obj] : ver.tr = t
   IN   ∧ BeginRd(t, obj)
           if t has written to obj, there's no anti-dependency
        ∧ rds′ = IF localWriteToObj THEN rds ELSE [rds EXCEPT ![obj] = @ ∪ {t}]
        ∧ inc′ = IF isActiveWrite THEN inc ∪ {tw} ELSE inc
        ∧ outc′ = IF isActiveWrite THEN outc ∪ {t} ELSE outc


True when transaction t creates a pivot transaction when reading obj

From Cahill et al.: for each version (xNew) of x that is newer than what T read:
  if xNew.creator is committed and xNew.creator.outConflict:
    abort(T)
ReadCreatesPivot(t, obj) ≜
   LET vr ≜ GetVer(obj, vis[t])
   IN   ∧ vr.tr ≠ t  reading our own write cannot create a pivot
        ∧ ∃ vw ∈ db[obj] : ∧ Concurrent(t, vw.tr)
                           ∧ tstate[vw.tr] = Committed
                           ∧ vw.tr ∈ outc

AbortRdS(t, obj) ≜
      ∧ op[t] = "r"
      ∧ rval[t] = Busy
      ∧ arg[t] = obj
```

```
        $\land\ ReadCreatesPivot(t,\ obj)$
        $\land\ op' = [op\ \text{EXCEPT}\ ![t] = \text{“a”}]$
        $\land\ arg' = [arg\ \text{EXCEPT}\ ![t] = \langle\rangle]$
        $\land\ rval' = [rval\ \text{EXCEPT}\ ![t] = Err]$
        $\land\ tr' = t$
        $\land\ tstate' = [tstate\ \text{EXCEPT}\ ![t] = Aborted]$
        $\land\ \text{UNCHANGED}\ \langle db,\ vis,\ tid,\ deadlocked,\ rds,\ inc,\ outc\rangle$
```

object version $v1$ is newer than object version $v2$

```
$Newer(v1,\ v2) \triangleq tid[v1.tr] > tid[v2.tr]$

$EndRdS(t,\ obj,\ val) \triangleq$
   LET $ver \triangleq GetVer(obj,\ vis[t])$
       $Ab(w) \triangleq w.tr = Aborted$
       $newer \triangleq \text{IF}\ ver.tr \neq t\ \text{THEN}\ \{w \in db[obj] : Newer(w,\ ver) \land \neg Ab(w)\}\ \text{ELSE}\ \{\}$
   IN
      $\land\ EndRd(t,\ obj,\ val)$
      $\land\ \neg ReadCreatesPivot(t,\ obj)$
       each later transaction that wrote has an inbound conflict
      $\land\ inc' = inc \cup \{w.tr : w \in newer\}$
       if there are any newer versions, $t$ has an outbound conflict
      $\land\ outc' = \text{IF}\ newer = \{\}\ \text{THEN}\ outc\ \text{ELSE}\ outc \cup \{t\}$
      $\land\ \text{UNCHANGED}\ rds$
```

True when transaction t creates a pivot transaction when reading obj

From *Cahill* et al.: if there is a *SIREAD lock*(rl) on x
  with $rl.owner$ is running or $commit(rl.owner) > begin(T)$:
    if $rl.owner$ is committed and $rl.owner.inConflict$:
      $abort(T)$

```
$WriteCreatesPivot(t,\ obj) \triangleq$
      $\exists\ tt \in rds[obj] \setminus \{t\} :$
        $\land\ \lor\ tstate[tt] = Open$
           $\lor\ tstate[tt] = Committed \land Concurrent(t,\ tt)$
        $\land\ tt \in inc$

$AbortWrS(t,\ obj) \triangleq$
   $\land\ \lor\ AbortWr(t,\ obj)$
     $\lor\ \land\ op[t] = \text{“w”}$
        $\land\ rval[t] = Busy$
        $\land\ WriteCreatesPivot(t,\ obj)$
        $\land\ op' = [op\ \text{EXCEPT}\ ![t] = \text{“a”}]$
        $\land\ arg' = [arg\ \text{EXCEPT}\ ![t] = \langle\rangle]$
        $\land\ rval' = [rval\ \text{EXCEPT}\ ![t] = Err]$
        $\land\ tr' = t$
        $\land\ tstate' = [tstate\ \text{EXCEPT}\ ![t] = Aborted]$
        ∧ UNCHANGED ⟨db, deadlocked, tid, vis⟩
    ∧ UNCHANGED ⟨rds, inc, outc⟩

EndWrS(t, obj, val) ≜
             active transactions
    LET     active ≜ {u ∈ Tr \ {t} : tstate[u] = Open}
             active transactions that are reading obj
            ards ≜ rds[obj] ∩ active
    IN    ∧ EndWr(t, obj, val)
          ∧ ¬WriteCreatesPivot(t, obj)
          ∧ outc′ = outc ∪ ards
          ∧ inc′ = IF ards = {} THEN inc ELSE inc ∪ {t}
          ∧ UNCHANGED rds

BeginCommit(t) ≜
    ∧ tstate[t] = Open
    ∧ rval[t] ≠ Busy
    ∧ op′ = [op EXCEPT ![t] = "c"]
    ∧ arg′ = [arg EXCEPT ![t] = ⟨⟩]
    ∧ rval′ = [rval EXCEPT ![t] = Busy]
    ∧ tr′ = t
    ∧ UNCHANGED ⟨db, vis, tid, deadlocked, tstate, rds, outc, inc⟩

Abort if commit would create a pivot transaction.
AbortCommit(t) ≜
    ∧ op[t] = "c"
    ∧ rval[t] = Busy
    ∧ t ∈ inc ∩ outc  pivot check
    ∧ op′ = [op EXCEPT ![t] = "a"]
    ∧ arg′ = [arg EXCEPT ![t] = ⟨⟩]
    ∧ rval′ = [rval EXCEPT ![t] = Err]
    ∧ tr′ = t
    ∧ tstate′ = [tstate EXCEPT ![t] = Aborted]
    ∧ UNCHANGED ⟨db, vis, tid, deadlocked, rds, outc, inc⟩

EndCommit(t) ≜
    ∧ op[t] = "c"
    ∧ rval[t] = Busy
    ∧ t ∉ inc ∩ outc
    ∧ op′ = [op EXCEPT ![t] = "c"]
    ∧ arg′ = [arg EXCEPT ![t] = ⟨⟩]
    ∧ rval′ = [rval EXCEPT ![t] = Ok]
    ∧ tr′ = t
    ∧ tstate′ = [tstate EXCEPT ![t] = Committed]
    ∧ UNCHANGED ⟨db, vis, tid, deadlocked, rds, outc, inc⟩
```

```
$BeginWrS(t, obj, val) \triangleq$
    $\land BeginWr(t, obj, val)$
    $\land \text{UNCHANGED } \langle rds, outc, inc \rangle$

$AbortS(t) \triangleq Abort(t) \land \text{UNCHANGED } \langle rds, outc, inc \rangle$
$DetectDeadlockS \triangleq DetectDeadlock \land \text{UNCHANGED } \langle rds, outc, inc \rangle$
$TerminationS \triangleq Termination \land \text{UNCHANGED } \langle rds, outc, inc \rangle$
$StartTransactionS(t) \triangleq StartTransaction(t) \land \text{UNCHANGED } \langle rds, outc, inc \rangle$

$NextS \triangleq$ $\lor \exists\, t \in Tr, obj \in Obj, val \in Val :$
                  $\lor StartTransactionS(t)$
                  $\lor BeginRdS(t, obj)$
                  $\lor AbortRdS(t, obj)$
                  $\lor EndRdS(t, obj, val)$
                  $\lor BeginWrS(t, obj, val)$
                  $\lor AbortWrS(t, obj)$
                  $\lor EndWrS(t, obj, val)$
                  $\lor BeginCommit(t)$
                  $\lor AbortCommit(t)$
                  $\lor EndCommit(t)$
                  $\lor AbortS(t)$
              $\lor DetectDeadlockS$
              $\lor TerminationS$

$vS \triangleq \langle op, arg, rval, tr, db, tstate, tid, vis, deadlocked, rds, inc, outc \rangle$

$LS \triangleq$ $\land \mathrm{WF}_{vS}(\exists\, t \in Tr :$ $\lor StartTransactionS(t)$
                                $\lor AbortCommit(t)$
                                $\lor EndCommit(t)$
                                $\lor AbortS(t))$
        $\land \mathrm{WF}_{vS}(\exists\, t \in Tr, obj \in Obj :$
              $\lor AbortRdS(t, obj)$
              $\lor AbortWrS(t, obj))$
        $\land \mathrm{WF}_{vS}(\exists\, t \in Tr, obj \in Obj, val \in Val :$
              $\lor EndRdS(t, obj, val)$
              $\lor EndWrS(t, obj, val))$
        $\land \mathrm{WF}_{vS}(DetectDeadlockS)$
        $\land \mathrm{SF}_{vS}(\exists\, t \in Tr : BeginCommit(t) \lor AbortS(t))$

$SpecS \triangleq InitS \land \Box[NextS]_{vS} \land LS$
```