MODULE $MVCC\_Ledger$

Middle level specification of *DLT Ledger*, expressed as a single state machine with *MVCC* validation

EXTENDS $Sequences,\ SequenceTheorems,\ Integers,\ TLAPS,\ Datatype$

Read-write set, which is a result of a simulation

CONSTANTS $RWSet$

State variables of this module

VARIABLES $state,$ current state of the ledger state machine.
$\quad chain,$ blockchain, a list of received transactions.
$\quad index$ unprocessed $TX$ index at the blockchain.

$vars \triangleq \langle state,\ chain,\ index \rangle$

Type invariant

At this module, endorsement is just a *RWSet*, which will be extended at lower models.

$Endorsement \triangleq RWSet$

each entry of blockchain now has a *RWSet*.

$ChainEntry \triangleq [tx : TX,\ endorsement : Endorsement,\ is\_valid : \text{BOOLEAN} \cup \{NULL\}]$

$Chain \triangleq Seq(ChainEntry)$

$TypeInv \triangleq$
$\quad \land\ state \in State$
$\quad \land\ index \in Nat$
$\quad \land\ index > 0$
$\quad$ Each $TX$ in the blockchain has a flag if it's valid or not. Before the $TX$ is processed, its value is $NULL$.
$\quad \land\ chain \in Chain$

---

Initial condition

$Init \triangleq$
$\quad \land\ state = InitState$ state is at the initial state, and
$\quad \land\ index = 1$
$\quad \land\ chain = \langle \rangle$ empty transaction queue.

---

Actions

(non-deterministic) simulaton result for the operation $f$

CONSTANT $SameOnRSet(\_,\ \_),\ Commit(\_,\ \_)$

ASSUME $SameOnRSetAxiom \triangleq \forall s \in State,\ rwset \in RWSet : SameOnRSet(s,\ rwset) \in \text{BOOLEAN}$

ASSUME $CommitAxiom \triangleq \forall s \in State,\ rwset \in RWSet : Commit(s,\ rwset) \in State$

$simulate(s,\ f) \triangleq$ CHOOSE $rwset \in RWSet :$

```
    (∀ ss ∈ State : SameOnRSet(ss, rwset) ⇒ Commit(ss, rwset) ∈ f[s])
ASSUME simulateAxiom ≜ ∀ s ∈ State, f ∈ Operation :
    ∃ rwset ∈ RWSet :
      (∀ ss ∈ State : SameOnRSet(ss, rwset) ⇒ Commit(ss, rwset) ∈ f[s])
LEMMA L1 ≜ ∀ s ∈ State, f ∈ Operation : simulate(s, f) ∈ RWSet
PROOF
    ⟨1⟩ TAKE s ∈ State, f ∈ Operation
    ⟨1⟩ QED
        BY simulateAxiom DEF simulate

endorsement(tx) ≜ simulate(state, tx.f)
```

*SubmitTx*: Client appends a transaction and its simulation result to the transaction queue.

```
SubmitTX(tx) ≜
    LET
        end ≜ endorsement(tx)
    IN
        ∧ chain′ = Append(chain, [tx ↦ tx, endorsement ↦ end, is_valid ↦ NULL])
        ∧ UNCHANGED ⟨state, index⟩
```

*ProcessTx*: *Ledger* processes the oldest unprocessed *TX* and updates its state by committing *RWSet* of *f*

```
ProcessTX_OK ≜
    LET
        f ≜ chain[index].tx.f
        rwset ≜ chain[index].endorsement
    IN
         ∧ Len(chain) ≥ index
        ∧ index ∈ DOMAIN chain
        ∧ SameOnRSet(state, rwset)
        ∧ chain′ = [chain EXCEPT ![index].is_valid = TRUE]   update validity flag
        ∧ index′ = index + 1   increment the index.
        ∧ state′ = Commit(state, rwset)   perform non-deterministic state transition by rwset.

ProcessTX_ERR ≜
    LET
        f ≜ chain[index].tx.f
        rwset ≜ chain[index].endorsement
    IN
         ∧ Len(chain) ≥ index
        ∧ index ∈ DOMAIN chain
        ∧ ¬SameOnRSet(state, rwset)
        ∧ chain′ = [chain EXCEPT ![index].is_valid = FALSE]   see above.
        ∧ index′ = index + 1   see above.
        ∧ UNCHANGED state   state does not change due to invalid TX.
```

$Next \triangleq (\exists\, tx \in TX : SubmitTX(tx)) \lor ProcessTX\_OK \lor ProcessTX\_ERR$

Specification

$Spec \triangleq Init \land \Box[Next]_{vars}$

---

Invariants

$Finality \triangleq$ TRUE *TODO*
$Safety \triangleq Finality$

Invariant (safety) on the blockchain
$ChainInv \triangleq$
  *chain* = (processed part) + (unprocessed part)
  $\land\, \forall\, i \in 1\,..\,index - 1 : chain[i].is\_valid \in \text{BOOLEAN}$
  $\land\, \forall\, i \in \{i \in Nat : index \leq i\} \cap \text{DOMAIN } chain : chain[i].is\_valid = NULL$

$Inv \triangleq TypeInv \land ChainInv$

Invariant (safety) on the *MVCC Ledger*
THEOREM $LedgerInv \triangleq Spec \Rightarrow \Box Inv$
PROOF
  $\langle 1\rangle 1\ Init \Rightarrow Inv$
    BY $InitStateAxiom$ DEF $Init, Inv, TypeInv, ChainInv, Chain$
  $\langle 1\rangle 2\ Inv \land [Next]_{vars} \Rightarrow Inv'$
   $\langle 2\rangle 1$ SUFFICES ASSUME $TypeInv, ChainInv, [Next]_{vars}$ PROVE $Inv'$ BY DEF $Inv$
   $\langle 2\rangle 2$ CASE $Next$
    $\langle 3\rangle$ USE DEF $Inv, Next$
    $\langle 3\rangle$ USE DEF $TypeInv, ChainInv, Chain, ChainEntry$
    $\langle 3\rangle 1$ CASE $(\exists\, tx \in TX : SubmitTX(tx))$
     $\langle 4\rangle$ USE DEF $SubmitTX$
     $\langle 4\rangle$a $\forall\, i \in \text{DOMAIN } chain : chain[i] = chain'[i]$ BY $\langle 3\rangle 1$
     $\langle 4\rangle 1\ TypeInv'$ BY $\langle 2\rangle 1, \langle 3\rangle 1, L1$ DEF $endorsement, Endorsement$
     $\langle 4\rangle 2\ ChainInv'$
      $\langle 5\rangle 1\ ChainInv!1'$ OBVIOUS
      $\langle 5\rangle 2\ ChainInv!2'$
       $\langle 6\rangle$a DOMAIN $chain' =$ DOMAIN $chain \cup \{Len(chain) + 1\}$ BY $TypeInv, \langle 3\rangle 1$
       $\langle 6\rangle 1$ PICK $tx \in TX : SubmitTX(tx)$ BY $\langle 3\rangle 1$
       $\langle 6\rangle 2$ TAKE $i \in (\{i \in Nat : index \leq i\} \cap \text{DOMAIN } chain)'$
       $\langle 6\rangle 3$ CASE $i \in (\{j \in Nat : index \leq j\} \cap \{Len(chain) + 1\})$ BY $\langle 2\rangle 1, \langle 4\rangle$a$, \langle 6\rangle 1, \langle 6\rangle 3$
       $\langle 6\rangle 4$ CASE $i \in (\{j \in Nat : index \leq j\} \cap \text{DOMAIN } chain)$ BY $\langle 2\rangle 1, \langle 4\rangle$a$, \langle 6\rangle 1, \langle 6\rangle 4$
       $\langle 6\rangle$ QED BY $\langle 2\rangle 1, \langle 6\rangle$a$, \langle 6\rangle 1, \langle 6\rangle 2, \langle 6\rangle 3, \langle 6\rangle 4$
      $\langle 5\rangle$ QED BY $\langle 5\rangle 1, \langle 5\rangle 2$
     $\langle 4\rangle$ QED BY $\langle 4\rangle 1, \langle 4\rangle 2$
    $\langle 3\rangle 2$ CASE $ProcessTX\_OK$
     $\langle 4\rangle$ USE DEF $ProcessTX\_OK$
     $\langle 4\rangle 1\ TypeInv'$ BY $\langle 2\rangle 1, \langle 3\rangle 2$ DEF $TX, Operation, TotalFunc$

```
                ⟨4⟩2 ChainInv′
                    ⟨5⟩ ChainInv!1′OBVIOUS
                    ⟨5⟩ ChainInv!2′BY ⟨2⟩1, ⟨3⟩2
                    ⟨5⟩ QED OBVIOUS
                ⟨4⟩ QED BY ⟨4⟩1, ⟨4⟩2
            ⟨3⟩3CASE ProcessTX_ERR
                ⟨4⟩ USE  DEF ProcessTX_ERR
                ⟨4⟩1 TypeInv′BY ⟨2⟩1, ⟨2⟩2, ⟨3⟩3  DEF TX, Operation, TotalFunc
                ⟨4⟩2 ChainInv′
                    ⟨5⟩ ChainInv!1′OBVIOUS
                    ⟨5⟩ ChainInv!2′BY ⟨2⟩1, ⟨3⟩3
                    ⟨5⟩ QED OBVIOUS
                ⟨4⟩ QED BY ⟨4⟩1, ⟨4⟩2
            ⟨3⟩ QED
                BY ⟨2⟩1, ⟨2⟩2, ⟨3⟩1, ⟨3⟩2, ⟨3⟩3
        ⟨2⟩3CASE UNCHANGED vars
            BY ⟨2⟩1, ⟨2⟩3  DEF Inv, TypeInv, ChainInv, vars
        ⟨2⟩ QED BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩3
    ⟨1⟩ QED BY PTL, ⟨1⟩1, ⟨1⟩2  DEF Spec

\* ideal form THEOREM  Spec ⇒ ∃ ex_state, ex_txs: LedgerSpec(ex_state, ex_txs)!Spec
  OMITTED

Refinement Mapping

VARIABLES h_state, h_chain,
          h_index

fmap(f(_), seq) ≜ [i ∈ DOMAIN seq ↦ f(seq[i])]
proj(r) ≜ [tx ↦ r.tx, is_valid ↦ r.is_valid]
Proj(seq) ≜ fmap(proj, seq)

h_state ≜ state
h_chain ≜ Proj(chain)
h_index ≜ index
h_vars ≜ ⟨h_state, h_chain, h_index⟩

HSpec ≜
    INSTANCE Ledger WITH state ← h_state, chain ← h_chain, index ← h_index

AXIOM NullEquality ≜ NULL = HSpec!NULL
LEMMA TypeEquality ≜ Operation = HSpec!Operation ∧ TX = HSpec!TX ∧ NULL = HSpec!NULL
PROOF
    BY NullEquality DEF TX, HSpec!TX, Operation, HSpec!Operation, TotalFunc, HSpec!TotalFunc

LEMMA fmapProperties ≜
    ASSUME
        NEW S, NEW T, NEW f(_),
        NEW seq ∈ Seq(S),
```

```
        ASSUME NEW e ∈ S PROVE f(e) ∈ T
    PROVE
        ∧ fmap(f, seq) ∈ Seq(T)
        ∧ Len(fmap(f, seq)) = Len(seq)
PROOF
    ⟨1⟩ DEFINE lhs ≜ fmap(f, seq)
    ⟨1⟩1. fmap(f, seq) ∈ Seq(T)
        ⟨2⟩1. ∀ i ∈ DOMAIN lhs : lhs[i] ∈ T
            ⟨3⟩ TAKE i ∈ DOMAIN lhs
            ⟨3⟩1. lhs[i] ∈ T BY DEF fmap
            ⟨3⟩ QED BY ⟨3⟩1
        ⟨2⟩ QED BY ⟨2⟩1, LenProperties, IsASeq DEF fmap
    ⟨1⟩2. Len(fmap(f, seq)) = Len(seq) BY DEF fmap
    ⟨1⟩ QED BY ⟨1⟩1, ⟨1⟩2

LEMMA projProperties ≜
    ASSUME NEW ce ∈ ChainEntry PROVE proj(ce) ∈ HSpec!ChainEntry
        BY TypeEquality DEF ChainEntry, HSpec!ChainEntry, proj

LEMMA ProjProperties ≜
    ASSUME NEW es ∈ Seq(ChainEntry)
    PROVE
        ∧ Proj(es) ∈ Seq(HSpec!ChainEntry)
        ∧ Len(es) = Len(Proj(es))
PROOF
    ⟨1⟩1. Proj(es) ∈ Seq(HSpec!ChainEntry)
        BY fmapProperties, projProperties DEF Proj
    ⟨1⟩2. Len(Proj(es)) = Len(es)
        BY fmapProperties, projProperties DEF Proj
    ⟨1⟩ QED BY ⟨1⟩1, ⟨1⟩2

LEMMA L2 ≜
    ∧ DOMAIN chain = DOMAIN h_chain
    ∧ ∀ i ∈ DOMAIN h_chain :
        h_chain[i].tx = chain[i].tx ∧ h_chain[i].is_valid = chain[i].is_valid
    BY DEF h_chain, proj

LEMMA L3 ≜
    ASSUME
        NEW S, NEW T, NEW f(_),
        NEW e ∈ S, NEW seq ∈ Seq(S),
        ASSUME NEW e0 ∈ S PROVE f(e0) ∈ T
    PROVE fmap(f, Append(seq, e)) = Append(fmap(f, seq), f(e))
PROOF
    ⟨1⟩1. DEFINE lhs ≜ fmap(f, Append(seq, e))
    ⟨1⟩2. DEFINE rhs ≜ Append(fmap(f, seq), f(e))
```

```
    ⟨1⟩3. lhs ∈ Seq(T) BY DEF fmap
    ⟨1⟩4. rhs ∈ Seq(T) BY fmapProperties
    ⟨1⟩5. Len(fmap(f, seq)) = Len(seq) BY DEF fmap
    ⟨1⟩6. Len(rhs) = Len(seq) + 1 BY ⟨1⟩5
    ⟨1⟩7. Len(lhs) = Len(seq) + 1 BY ⟨1⟩5 DEF fmap
    ⟨1⟩8. Len(lhs) = Len(rhs) BY ⟨1⟩6, ⟨1⟩7
    ⟨1⟩ HIDE DEF lhs, rhs
    ⟨1⟩9. ∀ i ∈ 1 .. Len(lhs) : lhs[i] = rhs[i]
        ⟨2⟩1. TAKE i ∈ 1 .. Len(lhs)
        ⟨2⟩2. lhs[i] = f(Append(seq, e)[i]) BY DEF lhs, fmap
        ⟨2⟩3. CASE i ∈ 1 .. Len(seq)
            ⟨3⟩1. f(Append(seq, e)[i]) = f(seq[i]) BY ⟨2⟩3 DEF lhs
            ⟨3⟩2. rhs[i] = fmap(f, seq)[i]
                BY AppendProperties, ⟨2⟩3, i ∈ 1 .. Len(fmap(f, seq)) DEF rhs, fmap
            ⟨3⟩3. fmap(f, seq)[i] = f(seq[i]) BY ⟨2⟩3, LenProperties, i ∈ DOMAIN seq DEF fmap
            ⟨3⟩ QED BY ⟨2⟩2, ⟨3⟩1, ⟨3⟩2, ⟨3⟩3 DEF lhs, rhs, fmap
        ⟨2⟩4. CASE i = Len(seq) + 1
            ⟨3⟩1. f(Append(seq, e)[i]) = f(e) BY ⟨2⟩4, AppendProperties
            ⟨3⟩2. i = Len(fmap(f, seq)) + 1 BY ⟨1⟩5, ⟨2⟩4, AppendProperties DEF rhs
            ⟨3⟩3. rhs[i] = f(e) BY ⟨3⟩2, AppendProperties DEF rhs, fmap
            ⟨3⟩ QED BY ⟨2⟩2, ⟨3⟩1, ⟨3⟩3 DEF lhs, rhs
        ⟨2⟩ QED BY ⟨1⟩7, ⟨2⟩3, ⟨2⟩4
    ⟨1⟩ QED BY ⟨1⟩3, ⟨1⟩4, ⟨1⟩8, ⟨1⟩9, SeqEqual DEF lhs, rhs


LEMMA L4 ≜
    ASSUME
        NEW e ∈ ChainEntry,
        NEW seq ∈ Seq(ChainEntry)
    PROVE
        Proj(Append(seq, e)) = Append(Proj(seq), proj(e))
PROOF
    ⟨1⟩1 ASSUME
            NEW e1 ∈ ChainEntry,
            NEW seq1 ∈ Seq(ChainEntry),
            ASSUME NEW e0 ∈ ChainEntry PROVE proj(e0) ∈ HSpec!ChainEntry
         PROVE
            Proj(Append(seq1, e1)) = Append(Proj(seq1), proj(e1))
         BY projProperties, L3 DEF Proj
    ⟨1⟩ QED BY ⟨1⟩1, projProperties
```

**Refinement Theorem**

```
THEOREM Refinement ≜ Spec ⇒ HSpec!Spec
PROOF
    ⟨1⟩ USE DEF Spec, HSpec!Spec, vars, HSpec!vars, proj
```

```
init case
⟨1⟩1. Init ⇒ HSpec!Init
    BY DEF Init, HSpec!Init, h_state, h_chain, h_index, Proj, fmap
next step (progress)
⟨1⟩2. Next ⇒ HSpec!Next ∨ UNCHANGED HSpec!vars
      ⟨2⟩0. ASSUME Next PROVE TypeInv BY LedgerInv DEF PTL, Spec, Inv
    ⟨2⟩1.CASE ∃ tx ∈ TX : SubmitTX(tx)
        ⟨3⟩1. PICK tx0 ∈ TX : SubmitTX(tx0)BY ⟨2⟩1
        ⟨3⟩3. ∃ tx ∈ HSpec!TX : HSpec!SubmitTX(tx)
            ⟨4⟩ USE TypeEquality
            ⟨4⟩1. WITNESS tx0 ∈ HSpec!TX
            ⟨4⟩3. HSpec!SubmitTX(tx0)!1
                ⟨5⟩1. DEFINE e ≜ [tx ↦ tx0, endorsement ↦ endorsement(tx0), is_valid ↦ NULL]
                ⟨5⟩c. chain ∈ Seq(ChainEntry)OMITTED
                ⟨5⟩a. e ∈ ChainEntryOMITTED
                ⟨5⟩b. proj(e) = [tx ↦ tx0, is_valid ↦ HSpec!NULL]BY TypeEquality
                ⟨5⟩ HIDE DEF e, proj
                ⟨5⟩k. Proj(Append(chain, e)) = Append(Proj(chain), proj(e))
                    BY ⟨5⟩c, ⟨5⟩a, ⟨5⟩b, L4 DEF h_chain
                ⟨5⟩ QED BY ⟨3⟩1, Proj(chain)′ = Proj(Append(chain, e)), ⟨5⟩k, ⟨5⟩b DEF SubmitTX, h_ch
            ⟨4⟩ QED BY ⟨3⟩1, ⟨4⟩3 DEF SubmitTX, HSpec!SubmitTX, h_index, h_state
        ⟨3⟩ QED
            BY ⟨2⟩1, ⟨3⟩3 DEF HSpec!Next
    ⟨2⟩2. ProcessTX_OK ⇒ HSpec!Next ∨ UNCHANGED HSpec!vars
        ⟨3⟩1. ProcessTX_OK ⇒
         ∨ HSpec!ProcessTX_OK
         ∨ UNCHANGED HSpec!vars
            ⟨4⟩1. HSpec!ProcessTX_OKOMITTED
            ⟨4⟩ QED BY ⟨4⟩1
        ⟨3⟩2. QED
            BY ⟨3⟩1 DEF HSpec!Next
    ⟨2⟩3. ProcessTX_ERR ⇒ HSpec!Next ∨ UNCHANGED HSpec!vars
        ⟨3⟩ USE DEF HSpec!Next, ProcessTX_ERR, HSpec!ProcessTX_ERR
        ⟨3⟩1. ASSUME ProcessTX_ERRPROVE HSpec!ProcessTX_ERR
            ⟨4⟩1. h_index ∈ DOMAIN h_chain
                ⟨5⟩a. chain ∈ Seq(ChainEntry)OMITTED
                ⟨5⟩1. DOMAIN chain = DOMAIN Proj(chain)BY ⟨5⟩a, ProjProperties
                ⟨5⟩ QED BY ⟨3⟩1, ⟨5⟩1, ProjProperties DEF h_index, h_chain
            ⟨4⟩3. h_chain′ = [h_chain EXCEPT ![h_index].is_valid = FALSE]
                ⟨5⟩ DEFINE lhs ≜ Proj(chain)′
                ⟨5⟩ DEFINE rhs ≜ [Proj(chain) EXCEPT ![index] = [Proj(chain)[index] EXCEPT !.is_valid
                ⟨5⟩a. lhs ∈ Seq(ChainEntry)OMITTED
                ⟨5⟩b. rhs ∈ Seq(ChainEntry)OMITTED
                ⟨5⟩1. Len(lhs) = Len(rhs)OMITTED
                ⟨5⟩2. ∀ i ∈ 1 . . Len(lhs) : lhs[i] = rhs[i]OMITTED
```

```
            ⟨5⟩ QED BY ⟨5⟩a, ⟨5⟩b, ⟨5⟩1, ⟨5⟩2, SeqEqual DEF h_chain, h_index
          ⟨4⟩4. h_index' = h_index + 1BY ⟨3⟩1  DEF h_index
          ⟨4⟩5. UNCHANGED h_stateBY ⟨3⟩1  DEF h_state
          ⟨4⟩ QED BY ⟨4⟩1, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5
        ⟨3⟩ QED BY ⟨3⟩1
      ⟨2⟩ QED
        BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩3  DEF Next, HSpec!Next
    next step (stutter)
    ⟨1⟩3. UNCHANGED vars ⇒ UNCHANGED HSpec!vars
      BY  DEF h_state, h_chain, h_index
    ⟨1⟩4. QED
      BY PTL, ⟨1⟩1, ⟨1⟩2, ⟨1⟩3
```

* Modification History
* Last modified Sun *Jul* 21 22:54:30 *JST* 2019 by *shinsa*
* Created *Tue Jul* 02 01:10:01 *JST* 2019 by *shinsa*