MODULE *Contracts*

This spec captures the behaviour of commitment transactions on the two sides of a Lightning channel.

We model the various kinds of outputs a commitment transactions will have over its lifetime.

The state of the commitment transaction changes in reponse to the various actions like supersede, publish, etc are taken by parties.

We also do not deal with the communication protocol between nodes for creating and updating commitment transactions. This spec will only focuss on the various commitment transaction and their lifecycle in response to interaction between parties and the blockchain.

We ignore the details of how transactions are signed and just mark transactions as signed. This lets us focus on the specifying the behaviour of the commitment transactions without dealing with lower level complexities.

The model defines the intial balance from *alice* to bob. TLA+ will handle situations where channels are balanced and when all the balance is on the other side.

TODO: Add actions for closing channels. Currenly we only have support for breach *tx* and the corresponding breach remedy txs.

TODO: Add *HTLCs*.

EXTENDS *Integers*,
TLC,
Sequences,
FiniteSets

CONSTANTS
CSV, — The csv value to use in contracts
InitialBalance — Initial balances for *alice* and bob

Current channel contracts only ever have two parties

$Party \triangleq \{\text{“alice”}, \text{“bob”}\}$

For the first revocation we only need two keys per party

$NumKey \triangleq 2$

Set of all keys

$Key \triangleq \{\langle p, k \rangle : p \in Party, k \in 0 \,..\, NumKey\}$

Value to capture missing *CSV* in output

$NoCSV \triangleq \text{CHOOSE } c : c \notin 0 \,..\, CSV$

Multisig outputs without *CSV* encumberance

$MultiSig \triangleq Party \times Party \times \{NoCSV\}$

Multisig outputs with *CSV* encumberance

$MultiSigWithCSV \triangleq Party \times Party \times \{CSV\}$

$P2WKH$ outputs, without encumbrance

$P2WKH \triangleq Key$

Set of all signatures for all commit txs. The signature in real world is related to the commit transaction. However, we leave out this complication of how the signature is generated. If there is a signature by a key on a tx, it is assumed it is correctly signed as per bitcoin's requirements

$Sig \triangleq \{\langle p, k\rangle : p \in Party, k \in 0 \,..\, NumKey - 1\}$

Value to capture unsigned transactions

$NoSig \triangleq \text{CHOOSE } s : s \notin Sig$

$$
\begin{aligned}
CT \triangleq\ [&index \mapsto Int,\\
&multisig \mapsto MultiSigWithCSV,\ pk \mapsto P2WKH,\\
&local\_sig \mapsto Sig \cup \{NoSig\},\\
&remote\_sig \mapsto Sig \cup \{NoSig\},\\
&balance \mapsto 0 \,..\, InitialBalance]
\end{aligned}
$$

$$
\begin{aligned}
OnChainTx \triangleq\ [&party \mapsto Party,\\
&index \mapsto Int,\\
&height \mapsto Int]
\end{aligned}
$$

$NoSpendHeight \triangleq -1$

VARIABLES

- $alice\_cts$, — Commitment tx for *alice*
- bob_cts, — Commitment tx for bob
- $alice\_brs$, — Breach remedy transactions for *alice*
- bob_brs, — Breach remedy transactions for bob
- $mempool$, — The CT txs that have been broadcasted.
- $published$, — The CT that has been included in a block and confirmed.
- $index$,
- $chain\_height$

$$
\begin{aligned}
vars \triangleq \langle &alice\_cts, bob\_cts, alice\_brs, bob\_brs, mempool, published,\\
&chain\_height, index\rangle
\end{aligned}
$$

Helper function to get other party

$OtherParty(party) \triangleq \text{CHOOSE } p \in Party : p \neq party$

Create a commitment transaction given the party, index and key to use.

$$
\begin{aligned}
&CreateCT(party, i, key\_num, balance) \triangleq\\
&\quad [party \mapsto party,\\
&\quad\ index \mapsto i,
\end{aligned}
$$

$multisig \mapsto \langle party,\ OtherParty(party),\ CSV \rangle,$
$pk \mapsto \langle party,\ key\_num \rangle,$
$local\_sig \mapsto NoSig,$
$remote\_sig \mapsto \langle OtherParty(party),\ key\_num \rangle,$
$balance \mapsto balance]$

$CreateOnChainTx(party,\ ix,\ height) \ \triangleq$
$[party \mapsto party,$
$height \ \mapsto height,$
$index \mapsto ix]$

$Init \ \triangleq$

Balanced channel to start with

$\land\ alice\_cts = \{CreateCT(\text{“alice”},\ 0,\ 0,\ InitialBalance)\}$
$\land\ bob\_cts = \{CreateCT(\text{“bob”},\ 0,\ 0,\ InitialBalance)\}$
$\land\ alice\_brs = \{\}$
$\land\ bob\_brs \ \ = \{\}$
$\land\ mempool = \{\}$
$\land\ published = \{\}$
$\land\ index = 1$
$\land\ chain\_height = 1$ The genesis block is the FT

$TypeInvariant \ \triangleq$
$\land\ \forall\ ct \in\ alice\_cts \cup bob\_cts \cup mempool :$
$\quad \land\ ct.party \ \in\ Party$
$\quad \land\ ct.index \ \in\ Nat$
$\quad \land\ ct.local\_sig \ \in\ Sig \cup \{NoSig\}$
$\quad \land\ ct.remote\_sig \ \in\ Sig \cup \{NoSig\}$
$\quad \land\ ct.pk \ \in\ P2WKH$
$\quad \land\ ct.multisig \ \in\ MultiSigWithCSV$
$\land\ \forall\ br \in\ alice\_brs \cup bob\_brs :$
$\quad \land\ br.index \ \in\ Nat$
$\quad \land\ br.pk \ \in\ P2WKH$
$\land\ \forall\ p \in\ published :$
$\quad \land\ p.party \ \in\ Party$
$\quad \land\ p.index \ \in\ Int$
$\quad \land\ p.height \ \in\ Int$
$\land\ index \ \in\ Nat$

$MaxIndex(party\_cts) \ \triangleq$
$(\text{CHOOSE}\ x \in\ party\_cts : \forall\ y \in\ party\_cts : x.index \geq y.index).index$

$LastCT(party\_cts) \ \triangleq$
$\text{CHOOSE}\ ct \in\ party\_cts : \forall\ y \in\ party\_cts : ct.index \geq y.index$

$AnyCT \triangleq (\text{CHOOSE } ct \in alice\_cts \cup bob\_cts : \text{TRUE})$

Create commitment transaction as well as the corresponding beach remedy txs.

Breach remedy transactions are pre-signed transactions instead of they private key being sent over to the other party.

delta is the balance going from *alice* to bob. We allow negative balances to enable payments in other other direction.

Parties are free to keep creating CT even if FT is spent. They will not be usable, but the protocol does not disallow this.

$$
\begin{array}{l}
SupersedeCommitmentTx(delta) \triangleq \\
\quad \land \\
\qquad \text{LET} \\
\qquad\quad key\_index \triangleq 1 \\
\qquad\quad last\_alice\_ct \triangleq LastCT(alice\_cts) \\
\qquad\quad last\_bob\_ct \triangleq LastCT(bob\_cts) \\
\qquad \text{IN}
\end{array}
$$

Create CTs till channel is not closed

$$
\begin{array}{l}
\qquad\quad \land published = \{\} \\
\qquad\quad \land last\_alice\_ct.balance + delta > 0 \\
\qquad\quad \land last\_bob\_ct.balance - delta > 0 \\
\qquad\quad \land alice\_cts' = alice\_cts \cup \\
\qquad\qquad\quad \{CreateCT(\text{“alice”}, index, key\_index, \\
\qquad\qquad\qquad last\_alice\_ct.balance + delta)\} \\
\qquad\quad \land bob\_cts' = bob\_cts \cup \\
\qquad\qquad\quad \{CreateCT(\text{“bob”}, index, key\_index, \\
\qquad\qquad\qquad last\_alice\_ct.balance - delta)\} \\
\qquad\quad \land alice\_brs' = alice\_brs \cup \\
\qquad\qquad\quad \{[index \mapsto index, pk \mapsto \langle \text{“bob”}, key\_index\rangle]\} \\
\qquad\quad \land bob\_brs' = bob\_brs \cup \\
\qquad\qquad\quad \{[index \mapsto index, pk \mapsto \langle \text{“alice”}, key\_index\rangle]\} \\
\qquad\quad \land index' = index + 1 \\
\quad \land \text{UNCHANGED } \langle mempool, published, chain\_height\rangle
\end{array}
$$

Broadcast a commitment transaction to the blockchain. The commitment is first signed. The protocol allows all commitments to be broadcast, what happens next depends on the status of the commitment transaction.

If the *tx* is the latest commitment transaction it can be spent later.

If not, it gives the other party a chance to spend the breach remedy *tx*.

TODO: We only spec CSV (self) commitment transaction. We need to handle the non-CSV output being published and co-op closes.

$$
\begin{array}{l}
BroadcastCommitment(party) \triangleq \\
\quad \land alice\_cts \neq \{\} \\
\quad \land bob\_cts \neq \{\} \\
\quad \land
\end{array}
$$

LET

$cts \triangleq$ IF $party = $ "alice" THEN $alice\_cts$ ELSE bob_cts

$ct \triangleq$ CHOOSE $ct \in cts$: TRUE

IN

$\land\ ct \notin mempool$

$\land\ mempool' = mempool \cup \{ct\}$

$\land$ UNCHANGED $\langle alice\_cts, bob\_cts, alice\_brs, bob\_brs, published, index, chain\_height\rangle$

Publish any transaction from *mempool* − this indeed is sparta. Any *mempool tx* can be confirmed. So we model just that. The only rule is to make sure the *CSV* has expired, and that is handled at the time of inserting the *tx* into *mempool*

$ConfirmMempoolTx \triangleq$

$\exists\, tx \in mempool$:

$\land\ chain\_height' = chain\_height + 1$

$\land\ published' = published \cup \{CreateOnChainTx(tx.party, tx.index, chain\_height')\}$

$\land\ mempool' = mempool \setminus \{tx\}$

$\land$ UNCHANGED $\langle alice\_cts, bob\_cts, alice\_brs, bob\_brs, index\rangle$

Publish a breach remedy transaction in response to a commitment transaction.

party is publishing the breach remedy *tx* when it is on index *CT*, and the chain is on height.

This *tx* is immediately published on chain.

TODO: We skip the *BR* going through the *mempool* and confirm it immediately. This can be improved too.

$BroadcastBR(party) \triangleq$

$\land$

LET

$cts \triangleq$ IF $party = $ "alice" THEN $alice\_cts$ ELSE bob_cts

IN

$\exists\, in\_mempool \in mempool$:

CT was broadcast by the other party

$\land\ in\_mempool.party = OtherParty(party)$

Revoked *CT* was broadcast

$\land\ in\_mempool.index < MaxIndex(cts)$

party already signed the ct as remote sig

$\land\ in\_mempool.remote\_sig[1] = party$

CSV hasn t expired - given *FT* is at height 1

$\land\ chain\_height < CSV$

$\land\ mempool' = mempool \cup \{in\_mempool\}$

$\land$ UNCHANGED $\langle alice\_cts, bob\_cts, alice\_brs, bob\_brs, mempool, index, published, chain\_height\rangle$

$Next \triangleq$

$\lor \exists\, d \in \{-1, 1\} : SupersedeCommitmentTx(d)$

$\lor \exists\, p \in Party : BroadcastCommitment(p)$

$\lor \exists\, p \in Party : BroadcastBR(p)$

$\lor\ ConfirmMempoolTx$

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

$Liveness \triangleq \exists\, p \in Party,\ d \in \{-1, 1\}:$

$\mathrm{WF\_}vars(PublishBR(p) \lor SupersedeCommitmentTx(d))$

$Liveness \triangleq \mathrm{WF}_{vars}(ConfirmMempoolTx)$

$FairSpec \triangleq Spec \land Liveness$