MODULE *LNContracsUsingBitcoinTransactions*

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 focus 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*,
*BitcoinTransactions*

CONSTANTS

| | |
|---|---|
| *INITIAL_BALANCE*, | Initial balances for alice and bob |
| *CHANNEL_PARTY* | Channel between parties |

VARIABLES

| | |
|---|---|
| *commitment_txs*, | Commitment txs held by each party. Not yet broadcast. |
| *breach_remedy_txs* | *BR* txs held by each party. Not yet broadcast. |

---

$SeqToSet(s) \triangleq \{s[i] : i \in \text{DOMAIN } s\}$

---

$vars \triangleq \langle chain\_height,\ transactions,\ mempool,\ published,\ commitment\_txs,\ breach\_remedy\_txs \rangle$

$Init \triangleq$
$\land\ transactions = [id \in TXID \mapsto [inputs \mapsto \langle\rangle,\ outputs \mapsto \langle\rangle]]$
$\land\ commitment\_txs = [p \in PARTY \mapsto \{\}]$
$\land\ breach\_remedy\_txs = [p \in PARTY \mapsto \{\}]$
$\land\ chain\_height = 0$

$\land\ mempool = \{\}$
$\land\ published = [id \in TXID \mapsto NoSpendHeight]$

$TypeOK \triangleq$
$\land\ transactions \in [TXID \to [inputs : Seq(Input),\ outputs : Seq(Output)]]$
$\land\ commitment\_txs \in [PARTY \to \text{SUBSET } TXID]$
$\land\ breach\_remedy\_txs \in [PARTY \to \text{SUBSET } TXID]$
$\land\ mempool \in \text{SUBSET } TXID$
$\land\ published \in [TXID \to Int]$

---

Choose keys for parties that have a channel.

The keys should have the same sequence number. This becomes important when parties create commitment transactions.

$ChooseChannelKeys \triangleq$
CHOOSE $\langle j, k \rangle \in Keys \times Keys :$
$\land\ \{j[1], k[1]\} \in CHANNEL\_PARTY$ Choose parties that have a channel
$\land\ j[2] = k[2]$ Choose keys with same index

$ChoosePartyKey(party) \triangleq$
CHOOSE $k \in Keys : k[1] = party$

$AllCommitmentsTxids \triangleq$
UNION $\{commitment\_txs[p] : p \in PARTY\}$

$AllBreachRemedyTxids \triangleq$
UNION $\{breach\_remedy\_txs[p] : p \in PARTY\}$

---

Confirm a transaction in *mempool*. This publishes the transaction.

We need to add a function like $IsOutputSpent(o)$ which checks if there is any transaction in published with $o$ as input.

$ConfirmTx(id) \triangleq$
$\land\ ConfirmMempoolTx(id)$
$\land$ UNCHANGED $\langle commitment\_txs, breach\_remedy\_txs \rangle$

We generate simple *p2wkh* transactions as inputs for funding transactions

Outputs both have $INITIAL\_BALANC * 2$ so that later we can have bi-directional channel with *INITIAL_BALANCE*

$CreateInputsForFundingTx(id, party) \triangleq$
$\land\ AddP2WKHCoinbaseToMempool(id, \langle ChoosePartyKey(party) \rangle, INITIAL\_BALANCE * 2)$
$\land$ UNCHANGED $\langle commitment\_txs, breach\_remedy\_txs \rangle$

**********

Create funding transaction that is signed by both parties for a channel.

```
**************************************************************************
AddFundingTxByPartyToMempool(id, channel) ≜
    ∃ o ∈ UnspentOutputs, p ∈ channel :
        transaction with id not created yet
        ∧ id ∉ mempool
        ∧ published[id] = NoSpendHeight
        ∧ id ∉ AllCommitmentsTxids
        ∧ id ∉ AllBreachRemedyTxids
        ∧ OutputOwnedByParty(o, p)
        ∧ LET fundingTx ≜ CreateMultisigTx(o, id, ChooseOutputKeys("multisig"), INITIAL_BALANCE)
          IN
            ∧ transactions' = [transactions EXCEPT ![id] = fundingTx]
            ∧ mempool' = mempool ∪ {id}
        ∧ UNCHANGED ⟨commitment_txs, breach_remedy_txs, chain_height, published⟩

 **************************************************************************
Create a commitment transaction for a party, sign it appropriately and send it to the other party.

Use a published funding transaction and its output as an input to the commitment tx.
**************************************************************************
 CreateCommitmentTxs(aid, bid) ≜
   ∃ ftxid ∈ DOMAIN published:
      ∧ published[ftxid] ≠ NoSpendHeight
      ∧ published[ftxid] < chain_height
      ∧ published[ftxid].outputs.type = "multisig"

──────────────────────────────────────────────────────────────────────────

Next ≜
    ∨ ∃ id ∈ TXID, party ∈ PARTY :
        ∨ CreateInputsForFundingTx(id, party)
    ∨ ∃ id ∈ TXID : ConfirmTx(id)
    ∨ ∃ id ∈ TXID, channel ∈ CHANNEL_PARTY :
        ∨ AddFundingTxByPartyToMempool(id, channel)
    ∨ ∃ id ∈ TXID: ConfirmTx(id)
    ∨ ∃ ⟨aid, bid⟩ ∈ TXID × TXID: CreateCommitmentTxs(aid, bid)

Spec ≜
    ∧ Init
    ∧ □[Next]⟨vars⟩

──────────────────────────────────────────────────────────────────────────
```