MODULE *LNContractsUsingBitcoinTransactions*

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

funding_txs,	The *TXID* for the channel funding txs
commitment_txs,	Commitment txs held by each party. Not yet broadcast.
breach_remedy_txs,	*BR* txs held by each party. Not yet broadcast.
funding_input_txs	Transactions spent by funding transactions

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

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

$Init \triangleq$
$\quad \wedge transactions = [id \in TXID \mapsto [inputs \mapsto \langle\rangle, outputs \mapsto \langle\rangle]]$

$\land\ funding\_txs = [channel \in CHANNEL\_PARTY \mapsto NoTxid]$
$\land\ funding\_input\_txs = [channel \in CHANNEL\_PARTY \mapsto NoTxid]$
$\land\ commitment\_txs = [channel \in CHANNEL\_PARTY \mapsto [p \in channel \mapsto NoTxid]]$
$\land\ breach\_remedy\_txs = [channel \in CHANNEL\_PARTY \mapsto [p \in channel \mapsto NoTxid]]$
$\land\ chain\_height = 0$
$\land\ mempool = \{\}$
$\land\ published = [id \in TXID \mapsto NoSpendHeight]$

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

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

$ChannelKeys(channel) \triangleq SetToSeq(\{\langle p, 1\rangle : p \in channel\})$

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

$OtherParty(channel, party) \triangleq$
CHOOSE $p \in channel : p \neq party$

$AllCommitmentsTxids \triangleq$
$\{commitment\_txs[cp[1]][cp[2]] :$
$cp \in \{\langle channel\_party, party\rangle \in CHANNEL\_PARTY \times PARTY : party \in channel\_party\}\}$

$AllBreachRemedyTxids \triangleq$
$\{breach\_remedy\_txs[cp[1]][cp[2]] :$
$cp \in \{\langle channel\_party, party\rangle \in CHANNEL\_PARTY \times PARTY : party \in channel\_party\}\}$

TXID id has not been used yet.

TODO: We might need to rethink how we track txids, but for now, this is the solution.

$IsUnused(id) \triangleq$
$\land\ id \notin mempool$

$\land\ published[id] = NoSpendHeight$
$\land\ id \notin AllCommitmentsTxids$
$\land\ id \notin AllBreachRemedyTxids$
$\land\ id \notin \{funding\_txs[t] : t \in CHANNEL\_PARTY\}$

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 $\langle commitment\_txs,\ breach\_remedy\_txs,\ funding\_txs,$
$funding\_input\_txs\rangle$

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

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

$CreateInputsForFundingTx(id,\ channel) \triangleq$
$\land\ funding\_txs[channel] = NoTxid$ No funding tx for channel exists
$\land\ funding\_input\_txs[channel] = NoTxid$
$\land\ \exists\, party \in channel :$
$\land$ LET $tx \triangleq [inputs \mapsto \langle\rangle,$
$outputs \mapsto \langle CreateP2WKHOutput(\langle ChoosePartyKey(party)\rangle,$
$INITIAL\_BALANCE * 2)\rangle]$
IN
$\land\ transactions' = [transactions$ EXCEPT $![id] = tx]$
$\land\ funding\_input\_txs' = [funding\_input\_txs$ EXCEPT $![channel] = id]$
$\land\ AddTxidToMempool(id)$
$\land$ UNCHANGED $\langle commitment\_txs,\ breach\_remedy\_txs,\ funding\_txs\rangle$

Create funding transaction. The funding tx is not locked at this point.

Once a commitment tx is signed then we can send this funding tx to the *mempool*.

Create a funding tx only if no funding tx exists for the channel

$CreateFundingTxByParty(id,\ channel) \triangleq$
$\exists\, o \in UnspentOutputs,\ p \in channel :$
transaction with id not created yet
$\land\ IsUnused(id)$
$\land\ funding\_txs[channel] = NoTxid$ No funding tx exists for the channel
$\land\ OutputOwnedByParty(o,\ p)$
$\land$ LET $fundingTx \triangleq CreateUnsignedMultisigTx(o,\ ChannelKeys(channel),\ INITIAL\_BALANCE)$
IN
$\land\ transactions' = [transactions$ EXCEPT $![id] = fundingTx]$
$\land\ funding\_txs' = [funding\_txs$ EXCEPT $![channel] = id]$

```
    ∧ UNCHANGED ⟨commitment_txs, breach_remedy_txs,
                 chain_height, published, mempool, funding_input_txs⟩

CreateCommitmentTxInput(ftxid, ftx) ≜
  ⟨[txid ↦ ftxid,
    index ↦ ftx.outputs[1].index,          Assume we only have one output in ftx
    sighash_flag ↦ "all",
    signed_by ↦ ftx.outputs[1].keys,
    hash_preimage ↦ NoHash]⟩

CreateCommitmentTxOutputs(channel, party, ftxid, ftx) ≜
  IF party = ftx.outputs[1].keys[1][1]
  THEN
    No output for other party as only one party funded the channel
    ⟨CreateP2WKHWithCSVOutput(⟨ftx.outputs[1].keys[1]⟩,
                              ftx.outputs[1].amount)⟩
  ELSE ⟨⟩

Create a commitment tx for for a channel parties.
CreateCommitmentTx(txid, channel, party) ≜
  txid is not used
  ∧ IsUnused(txid)
  Funding tx for channel exists
  ∧ funding_txs[channel] ≠ NoTxid
  Commitment tx for channel and party doesn't exist
  ∧ commitment_txs[channel][party] = NoTxid
  Create the commitment tx for party paying back to funder with CSV
  ∧ LET ftxid ≜ funding_txs[channel]
        ftx ≜ transactions[ftxid]
        ftx_input ≜ transactions[ftx.inputs[1].txid]
        Create commitment tx, that spends ftx and creates outputs for party and other party
        Use amounts based on party = funding input key holder
        tx ≜ [inputs ↦ CreateCommitmentTxInput(ftxid, ftx),
              outputs ↦ CreateCommitmentTxOutputs(channel, party, ftxid, ftx)]
    IN
      ∧ commitment_txs' = [commitment_txs EXCEPT ![channel][party] = txid]
      ∧ transactions' = [transactions EXCEPT ![txid] = tx]
  ∧ UNCHANGED ⟨breach_remedy_txs, funding_txs,
               chain_height, published, mempool, funding_input_txs⟩

Add funding tx to mempool once commitment transactions have been signed and shared.
AddFundingTxToMempool(txid, channel) ≜
  LET
    funding_txid ≜ funding_txs[channel]
  IN
```

$\land$ $funding\_txid \neq NoTxid$ A funding tx exists
$\land \forall\, p \in channel :$
There is a commitment tx for both parties
$\land$ $commitment\_txs[channel][p] \neq NoTxid$
both commitment tx spending the funding tx
$\land$ $transactions[commitment\_txs[channel][p]].inputs[1].txid = funding\_txid$
$\land$ $mempool' = mempool \cup \{funding\_txid\}$
$\land$ UNCHANGED $\langle commitment\_txs,\ breach\_remedy\_txs,$
$chain\_height,\ published,\ funding\_txs,\ transactions,$
$funding\_input\_txs\rangle$

$Next \triangleq$
$\lor \exists\, id \in TXID,\ channel \in CHANNEL\_PARTY :$
$\lor\ CreateInputsForFundingTx(id,\ channel)$
$\lor \exists\, id \in TXID : ConfirmTx(id)$
$\lor \exists\, id \in TXID,\ channel \in CHANNEL\_PARTY :$
$\lor\ CreateFundingTxByParty(id,\ channel)$
$\lor\ AddFundingTxToMempool(id,\ channel)$
$\lor \exists\, channel \in CHANNEL\_PARTY :$
$\exists\, \langle txid,\ party\rangle \in TXID \times channel :$
$CreateCommitmentTx(txid,\ channel,\ party)$

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