MODULE *SASwap*

SASwap TLA+ specification (c) by Dmitry Petukhov (https://github.com/dgpv)
This SASwap TLA+ specification is licensed under a Creative Commons
Attribution-ShareAlike 4.0 International License
<http://creativecommons.org/licenses/by-sa/4.0/>

EXTENDS *Naturals*, *Sequences*, *TLC*

CONSTANT *BLOCKS_PER_DAY*
ASSUME $BLOCKS\_PER\_DAY > 0$

CONSTANT *MAX_DAYS_TO_CONCLUDE*
ASSUME $MAX\_DAYS\_TO\_CONCLUDE > 4$

CONSTANT *STEALTHY_SEND_POSSIBLE*
ASSUME $STEALTHY\_SEND\_POSSIBLE \in \text{BOOLEAN}$

VARIABLES *blocks*, *block_txs*, *mempool*, *shared_knowledge*, *sigs*, *signers_map*

$sigState \triangleq \langle sigs, signers\_map \rangle$
$networkState \triangleq \langle blocks, block\_txs, mempool \rangle$
$fullState \triangleq \langle networkState, sigState, shared\_knowledge \rangle$

Can use this predicate to limit possible state space by
ignoring the states after *MAX_DAYS_TO_CONCLUDE* has passed
$ConcludedInFiniteDays \triangleq$
$Len(blocks) \leq BLOCKS\_PER\_DAY * MAX\_DAYS\_TO\_CONCLUDE$

Can use this predicate to limit possible state space by
ignoring the states where nothing was sent/confirmed in a day
$EachDaySomethingIsConfirmed \triangleq$
$\neg\exists\, bn \in \text{DOMAIN } blocks :$
$\forall\, bn\_next \in bn \,..\, bn + BLOCKS\_PER\_DAY - 1 :$
$\wedge\ bn\_next \in \text{DOMAIN } blocks$
$\wedge\ blocks[bn\_next] = \{\}$

Define unique values to avoid using strings everywhere

$Alice \triangleq$ "Alice"
$Bob \triangleq$ "Bob"

$sigAlice \triangleq$ "sigAlice"

$sigBob \triangleq$ "sigBob"
$secretAlice \triangleq$ "secretAlice"
$secretBob \triangleq$ "secretBob"

$tx\_start\_A \triangleq$ "tx_start_A"
$tx\_start\_B \triangleq$ "tx_start_B"
$tx\_success \triangleq$ "tx_success"
$tx\_refund\_1 \triangleq$ "tx_refund_1"
$tx\_revoke \triangleq$ "tx_revoke"
$tx\_refund\_2 \triangleq$ "tx_refund_2"
$tx\_timeout \triangleq$ "tx_timeout"

$tx\_spend\_B \triangleq$ "tx_spend_B"
$tx\_spend\_success \triangleq$ "tx_spend_success"
$tx\_spend\_refund\_1 \triangleq$ "tx_spend_refund_1"
$tx\_spend\_refund\_2 \triangleq$ "tx_spend_refund_2"
$tx\_spend\_timeout \triangleq$ "tx_spend_timeout"

$participants \triangleq \{Alice, Bob\}$

$all\_sigs \triangleq \{sigAlice, sigBob, secretAlice, secretBob\}$

$Counterparty(p) \triangleq \text{CHOOSE } c \in participants : c \neq p$

$ConfirmedTransactions \triangleq$
UNION $\{blocks[bn] : bn \in \text{DOMAIN } blocks\}$

$SentTransactions \triangleq ConfirmedTransactions \cup mempool$

$$\begin{aligned} dependency\_map \triangleq [&tx\_success \mapsto tx\_start\_A,\\ &tx\_refund\_1 \mapsto tx\_start\_A,\\ &tx\_revoke \mapsto tx\_start\_A,\\ &tx\_refund\_2 \mapsto tx\_revoke,\\ &tx\_timeout \mapsto tx\_revoke,\\ &tx\_spend\_B \mapsto tx\_start\_B,\\ &tx\_spend\_success \mapsto tx\_success,\\ &tx\_spend\_refund\_1 \mapsto tx\_refund\_1,\\ &tx\_spend\_refund\_2 \mapsto tx\_refund\_2,\\ &tx\_spend\_timeout \mapsto tx\_timeout] \end{aligned}$$

$all\_transactions \triangleq$

each transacton is either dependant or a dependency

$$\text{DOMAIN } dependency\_map \cup \{dependency\_map[x] : x \in \text{DOMAIN } dependency\_map\}$$

transactions that are mutually exclusive

$$conflicts \triangleq \{\{tx\_success,\ tx\_refund\_1,\ tx\_revoke\},\ \{tx\_refund\_2,\ tx\_timeout\}\}$$

transactions *Alice* initially shares signatures on

$$phase0\_to\_share\_Alice \triangleq \{tx\_revoke,\ tx\_timeout\}$$

transactions *Bob* initially shares signatures on

$$phase0\_to\_share\_Bob \triangleq \{tx\_refund\_1,\ tx\_revoke,\ tx\_refund\_2,\ tx\_timeout\}$$

$$HasDependencies(tx) \triangleq tx \in \text{DOMAIN } dependency\_map$$

$$DependencyConfirmed(tx) \triangleq dependency\_map[tx] \in ConfirmedTransactions$$

$$DependencySent(tx) \triangleq DependencyConfirmed(tx) \lor dependency\_map[tx] \in mempool$$

$$DependencyBlock(tx) \triangleq \text{CHOOSE } bn \in \text{DOMAIN } blocks : dependency\_map[tx] \in blocks[bn]$$

$$NoConflicts(tx) \triangleq \forall\, cfl\_set \in conflicts : \lor\, tx \notin cfl\_set \lor \neg\exists\, cfl\_tx \in cfl\_set \setminus \{tx\} : cfl\_tx \in SentTransactions$$

$$SharedSecrets \triangleq \{x[2] : x \in \{xx \in shared\_knowledge : xx[2] \in \{secretAlice,\ secretBob\}\}\}$$

Signatures currently available to the sender
includes *sigs* made by the sender themself,
and anything from shared knowledge

$$AvailableSigs(tx,\ sender) \triangleq sigs[sender][tx] \cup \{x[2] : x \in \{xx \in shared\_knowledge : xx[1] = tx\}\} \cup SharedSecrets$$

All signatures known for certain transaction.
These go to shared knowledge when the transaction is mined

(unless already shared)
This is a simplification, because in general there could be
unpublished signatures when threshold signing is used.
But for this contract, this is *OK*.

$$AllSigsForTx(tx) \triangleq \text{UNION } \{sigs[sender][tx] : sender \in participants\}$$

Adaptor signatures are modelled as having to supply 3 values
for the signature set, where two values are the *sigs*,
and one value is a secret.
For modelling purposes, the secret acts as just another sig.

$$\begin{array}{l}
SpendConditionsSatisfied(tx,\ sender) \triangleq \\
\quad \text{LET } HasSigs(ss) \triangleq ss \subseteq AvailableSigs(tx,\ sender) \\
\quad \text{IN} \quad \lor \land tx = tx_start_A \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice\}) \\
\qquad\qquad \lor \land tx = tx_start_B \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigBob\}) \\
\qquad\qquad \lor \land tx = tx_success \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice,\ sigBob,\ secretBob\}) \\
\qquad\qquad \lor \land tx = tx_refund_1 \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice,\ sigBob,\ secretAlice\}) \\
\qquad\qquad\quad\ \ \land Len(blocks) \geq BLOCKS_PER_DAY \\
\qquad\qquad \lor \land tx = tx_revoke \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice,\ sigBob\}) \\
\qquad\qquad\quad\ \ \land Len(blocks) \geq BLOCKS_PER_DAY * 2 \\
\qquad\qquad \lor \land tx = tx_refund_2 \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice,\ sigBob,\ secretAlice\}) \\
\qquad\qquad\quad\ \ \land DependencyConfirmed(tx) \\
\qquad\qquad\quad\ \ \land Len(blocks) \geq DependencyBlock(tx) + BLOCKS_PER_DAY \\
\qquad\qquad \lor \land tx = tx_timeout \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice,\ sigBob\}) \\
\qquad\qquad\quad\ \ \land DependencyConfirmed(tx) \\
\qquad\qquad\quad\ \ \land Len(blocks) \geq DependencyBlock(tx) + BLOCKS_PER_DAY * 2 \\
\qquad\qquad \lor \land tx = tx_spend_B \\
\qquad\qquad\quad\ \ \land HasSigs(\{secretAlice,\ secretBob\}) \\
\qquad\qquad \lor \land tx = tx_spend_success \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigBob\}) \\
\qquad\qquad \lor \land tx = tx_spend_refund_1
\end{array}$$

$\land\ HasSigs(\{sigAlice\})$
$\land\ DependencyConfirmed(tx)$
$\land\ Len(blocks) \geq DependencyBlock(tx) + BLOCKS\_PER\_DAY$
$\lor\ \land\ tx = tx\_spend\_refund\_2$
$\land\ HasSigs(\{sigAlice\})$
$\lor\ \land\ tx = tx\_spend\_timeout$
$\land\ HasSigs(\{sigBob\})$

Unclear what the result of this tx should be
$\lor\ \land\ tx = tx\_spend\_refund\_1\_cooperative$

$CanEnterMempool(tx,\ sender) \triangleq$
$\land\ tx \notin SentTransactions$
$\land\ NoConflicts(tx)$
$\land\ \lor\ HasDependencies(tx) \land DependencySent(tx)$
$\lor\ \neg HasDependencies(tx)$
$\land\ SpendConditionsSatisfied(tx,\ sender)$

$Share(knowledge) \triangleq shared\_knowledge' = shared\_knowledge \cup knowledge$

Note that the *sigs* record-of-records may not be the most elegant
data structure for the purpose, might be worth it to explore other options
$SignTx(tx,\ ss,\ signer) \triangleq$
$\land\ \forall\, s \in ss : s \in signers\_map[signer]$
$\land\ sigs' = [sigs\ \text{EXCEPT}$
$![signer] = [sigs[signer]\ \text{EXCEPT}$
$![tx] = sigs[signer][tx] \cup ss]]$

$SendTx(tx,\ sender) \triangleq$
$\land\ CanEnterMempool(tx,\ sender)$
$\land\ mempool' = mempool \cup \{tx\}$
$\land\ Share(\{\langle tx,\ s\rangle : s \in sigs[sender][tx]\})$

Give tx directly to miner, bypassing global *mempool*
$StealthySendTx(tx,\ sender) \triangleq$
$\land\ STEALTHY\_SEND\_POSSIBLE$
$\land\ CanEnterMempool(tx,\ sender)$
$\land\ block\_txs' = block\_txs \cup \{tx\}$

If participant has a transaction ready to be sent,

they can send it to *mempool*, or stealthy to the miner
(if *STEALTHY_SEND_POSSIBLE* is TRUE)

$$
\begin{array}{l}
SendSomething \triangleq \\
\quad \exists\, sender \in participants : \\
\quad\quad \exists\, tx \quad \in all\_transactions : \\
\quad\quad\quad \lor \land SendTx(tx,\, sender) \\
\quad\quad\quad\quad \land \text{UNCHANGED } block\_txs \\
\quad\quad\quad \lor \land StealthySendTx(tx,\, sender) \\
\quad\quad\quad\quad \land \text{UNCHANGED } \langle mempool,\, shared\_knowledge \rangle
\end{array}
$$

Participant actions

Conditions to divide the action into phases according to original spec

$$
\begin{array}{l}
Phase\_3\_cond \triangleq tx\_start\_B \in ConfirmedTransactions \\
Phase\_2\_cond \triangleq tx\_start\_A \in ConfirmedTransactions \\
Phase\_1\_cond \triangleq \\
\quad \land \forall\, tx \in phase0\_to\_share\_Alice : \\
\quad\quad \exists\, signed\_pair \in shared\_knowledge : signed\_pair = \langle tx,\, sigAlice \rangle \\
\quad \land \forall\, tx \in phase0\_to\_share\_Bob : \\
\quad\quad \exists\, signed\_pair \in shared\_knowledge : signed\_pair = \langle tx,\, sigBob \rangle
\end{array}
$$

$$
\begin{array}{l}
InPhase\_3 \triangleq \\
\quad \land Phase\_3\_cond
\end{array}
$$

$$
\begin{array}{l}
InPhase\_2 \triangleq \\
\quad \land Phase\_2\_cond \\
\quad \land \neg Phase\_3\_cond
\end{array}
$$

$$
\begin{array}{l}
InPhase\_1 \triangleq \\
\quad \land Phase\_1\_cond \\
\quad \land \neg Phase\_2\_cond \\
\quad \land \neg Phase\_3\_cond
\end{array}
$$

$$
\begin{array}{l}
InPhase\_0 \triangleq \\
\quad \land \neg Phase\_1\_cond \\
\quad \land \neg Phase\_2\_cond \\
\quad \land \neg Phase\_3\_cond
\end{array}
$$

helper operators to declutter the action expressions

$NoSigning \triangleq$ UNCHANGED $sigState$
$NothingShared \triangleq$ UNCHANGED $shared\_knowledge$

$AliceAction \triangleq$
LET $Sign(tx, ss) \triangleq SignTx(tx, ss, Alice)$
IN $\lor \land InPhase\_0$
$\land Share(\{\langle tx, sigAlice\rangle : tx \in phase0\_to\_share\_Alice\})$
$\land NoSigning$
$\lor \land InPhase\_1$
$\land Sign(tx\_start\_A, \{sigAlice\})$
$\land NothingShared$
$\lor \land InPhase\_2$
$\land$ Do nothing or refund, if timelock allows
$\lor NoSigning$
$\lor Sign(tx\_refund\_1, \{sigAlice, secretAlice\})$
$\land NothingShared$
$\lor \land InPhase\_3$
$\land \lor \land Share(\{\langle tx\_success, sigAlice\rangle\})$
$\land NoSigning$
$\lor$ Can revoke on short path until $tx\_success$ is shared
$\land \langle tx\_success, sigAlice\rangle \notin shared\_knowledge$
$\land Sign(tx\_refund\_1, \{sigAlice, secretAlice\})$
$\land NothingShared$
$\lor \land \lor Sign(tx\_spend\_B, \{secretAlice\})$
$\lor Sign(tx\_refund\_2, \{sigAlice, secretAlice\})$
$\lor Sign(tx\_spend\_refund\_1, \{sigAlice, secretAlice\})$
$\lor Sign(tx\_spend\_refund\_2, \{sigAlice, secretAlice\})$
$\land NothingShared$

$BobAction \triangleq$
LET $Sign(tx, ss) \triangleq SignTx(tx, ss, Bob)$
IN $\lor \land InPhase\_0$
$\land Share(\{\langle tx, sigBob\rangle : tx \in phase0\_to\_share\_Bob\})$
$\land NoSigning$
$\lor \land InPhase\_1$ No specific actions
$\land NoSigning$
$\land NothingShared$
$\lor \land InPhase\_2$

$$
\begin{array}{l}
\quad \land \mathit{Sign}(\mathit{tx\_start\_B},\ \{\mathit{sigBob}\}) \\
\quad \land \mathit{NothingShared} \\
\lor \ \land \mathit{InPhase\_3} \\
\quad \land \ \lor \mathit{Sign}(\mathit{tx\_spend\_B},\ \{\mathit{secretBob}\}) \\
\qquad \lor \mathit{Sign}(\mathit{tx\_success},\ \{\mathit{sigBob},\ \mathit{secretBob}\}) \\
\qquad \lor \mathit{Sign}(\mathit{tx\_spend\_success},\ \{\mathit{sigBob}\}) \\
\qquad \lor \mathit{Sign}(\mathit{tx\_timeout},\ \{\mathit{sigBob}\}) \\
\qquad \lor \mathit{Sign}(\mathit{tx\_spend\_timeout},\ \{\mathit{sigBob}\}) \\
\quad \land \mathit{NothingShared}
\end{array}
$$

$$
\begin{array}{l}
\mathit{IncludeTxIntoBlock} \ \triangleq \\
\quad \land \exists\, \mathit{tx} \in \mathit{mempool} : \\
\qquad \land \ \text{IF}\ \mathit{HasDependencies}(\mathit{tx}) \\
\qquad\quad \text{THEN}\ \ \lor \mathit{DependencyConfirmed}(\mathit{tx}) \\
\qquad\qquad\qquad \lor \mathit{dependency\_map}[\mathit{tx}] \in \mathit{block\_txs} \\
\qquad\quad \text{ELSE}\ \ \text{TRUE} \\
\qquad \land \mathit{block\_txs}' = \mathit{block\_txs} \cup \{\mathit{tx}\} \\
\quad \land \ \text{UNCHANGED}\ \langle \mathit{blocks},\ \mathit{mempool},\ \mathit{shared\_knowledge} \rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{MineTheBlock} \ \triangleq \\
\quad \land \mathit{blocks}' = \mathit{Append}(\mathit{blocks},\ \mathit{block\_txs}) \\
\quad \land \mathit{mempool}' = \mathit{mempool} \setminus \mathit{block\_txs} \\
\quad \land \mathit{Share}(\text{UNION}\ \{\{\langle \mathit{tx},\ s \rangle : s \in \mathit{AllSigsForTx}(\mathit{tx})\} : \\
\qquad\qquad\qquad\qquad \mathit{tx} \in \mathit{block\_txs} \setminus \mathit{mempool}\}) \\
\quad \land \mathit{block\_txs}' = \{\}
\end{array}
$$

$$
\mathit{MinerAction} \ \triangleq \ \mathit{IncludeTxIntoBlock} \lor \mathit{MineTheBlock}
$$

Invariants

$$
\begin{array}{l}
\mathit{TypeOK} \triangleq \\
\quad \land \quad \forall\, \mathit{pair} \in \mathit{shared\_knowledge} : \\
\qquad\quad \land\, \mathit{pair}[1] \in \mathit{all\_transactions} \\
\qquad\quad \land\, \mathit{pair}[2] \in \mathit{all\_sigs} \\
\quad \land \quad \text{DOMAIN } \mathit{sigs} = \mathit{participants} \\
\quad \land \quad \text{DOMAIN } \mathit{signers\_map} = \mathit{participants} \\
\quad \land \quad \forall\, \mathit{sender} \in \text{DOMAIN } \mathit{sigs} : \\
\qquad\quad \forall\, \mathit{tx} \in \text{DOMAIN } \mathit{sigs}[\mathit{sender}] : \\
\qquad\qquad \land\, \mathit{tx} \in \mathit{all\_transactions} \\
\qquad\qquad \land\, \mathit{sigs}[\mathit{sender}][\mathit{tx}] \subseteq \mathit{all\_sigs}
\end{array}
$$

$$
\begin{array}{l}
\mathit{ConsistentPhase} \triangleq \\
\quad \text{LET } \mathit{phases} \triangleq \langle \mathit{InPhase\_0},\, \mathit{InPhase\_1},\, \mathit{InPhase\_2},\, \mathit{InPhase\_3} \rangle \\
\quad \text{IN} \quad \land\, \exists\, i \in \text{DOMAIN } \mathit{phases} : \mathit{phases}[i] \\
\qquad\quad \land\, \forall\, i \in \text{DOMAIN } \mathit{phases} : \\
\qquad\qquad \text{IF } \mathit{phases}[i] \\
\qquad\qquad\quad \text{THEN } \neg \exists\, \mathit{ii} \in \text{DOMAIN } \mathit{phases} : \mathit{ii} \neq i \land \mathit{phases}[\mathit{ii}] \\
\qquad\qquad\quad \text{ELSE TRUE}
\end{array}
$$

$$
\begin{array}{l}
\mathit{NoConcurrentSecretKnowledge} \triangleq \\
\quad \lor\, \mathit{tx\_spend\_B} \in \mathit{SentTransactions} \\
\quad \lor\, \neg(\{\mathit{secretAlice},\, \mathit{secretBob}\} \subseteq \mathit{SharedSecrets})
\end{array}
$$

Can use this invariant to check if certain state can be reached
If the *CounterExample* invariant is violated, then the state
has been reached.

$$\mathit{CounterExample} \triangleq \text{TRUE} \land \ldots$$

Temporal properties, Not tested at the moment

$$
\begin{array}{l}
\mathit{AllEventuallyConfirmed} \triangleq \\
\quad \forall\, \mathit{tx} \in \mathit{all\_transactions} : \Diamond(\mathit{tx} \in \mathit{ConfirmedTransactions})
\end{array}
$$

$$
\begin{array}{l}
\mathit{SuccessTxLeadsToSpentB} \triangleq \\
\quad \mathit{tx\_success} \in \mathit{SentTransactions} \leadsto \mathit{tx\_spend\_B} \in \mathit{ConfirmedTransactions}
\end{array}
$$

High-level spec

$$
\begin{aligned}
&\mathit{Init} \triangleq \\
&\quad \land\ \mathit{blocks} = \langle\rangle \\
&\quad \land\ \mathit{block\_txs} = \{\} \\
&\quad \land\ \mathit{mempool} = \{\} \\
&\quad \land\ \mathit{shared\_knowledge} = \{\} \\
&\quad \land\ \mathit{sigs} = [\mathit{Alice} \mapsto [\mathit{tx} \in \mathit{all\_transactions} \mapsto \{\}], \\
&\qquad\qquad\ \mathit{Bob} \mapsto [\mathit{tx} \in \mathit{all\_transactions} \mapsto \{\}]] \\
&\quad \land\ \mathit{signers\_map} = [\mathit{Alice} \mapsto \{\mathit{sigAlice},\ \mathit{secretAlice}\}, \\
&\qquad\qquad\qquad\ \mathit{Bob}\ \mapsto \{\mathit{sigBob},\ \mathit{secretBob}\}]
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{Next} \triangleq \\
&\quad \land\ \mathit{ConcludedInFiniteDays} \\
&\quad \land\ \mathit{EachDaySomethingIsConfirmed} \\
&\qquad \text{Note that } \mathit{signers\_map} \text{ is always unchanged at the moment} \\
&\quad \land\ \lor\ \mathit{AliceAction} \quad \land \text{UNCHANGED } \langle \mathit{networkState},\ \mathit{signers\_map} \rangle \\
&\qquad \lor\ \mathit{BobAction} \quad \land \text{UNCHANGED } \langle \mathit{networkState},\ \mathit{signers\_map} \rangle \\
&\qquad \lor\ \mathit{SendSomething}\ \land \text{UNCHANGED } \langle \mathit{blocks},\ \mathit{sigState} \rangle \\
&\qquad \lor\ \mathit{MinerAction} \quad \land \text{UNCHANGED } \mathit{sigState}
\end{aligned}
$$

$$
\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{fullState}} \quad \land \text{WF\_}\mathit{networkState}(\mathit{Next})
$$