MODULE *SASwap*

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

CONSTANT $PARTICIPANTS\_IRRATIONAL$ Can participants act irrational ?
ASSUME $PARTICIPANTS\_IRRATIONAL \in$ BOOLEAN

CONSTANT $BLOCKS\_PER\_DAY$
More blocks per day means larger state space to check
ASSUME $BLOCKS\_PER\_DAY \geq 1$

A transaction that has no deadline can be 'stalling',
i.e. not being sent while being enabled, for this number of days
CONSTANT $MAX\_DAYS\_STALLING$
More days allowed stalling means larger state space to check
ASSUME $MAX\_DAYS\_STALLING \geq 1$

Is it possible for participants to send transactions
bypassing the *mempool* (give directly to the miner)
CONSTANT $STEALTHY\_SEND\_POSSIBLE$
When TRUE, the state space is increased dramatically.
ASSUME $STEALTHY\_SEND\_POSSIBLE \in$ BOOLEAN

Operator to create transaction instances
$Tx(id, ss, by, to, via) \triangleq$
$\quad [id \mapsto id, ss \mapsto ss, to \mapsto to, by \mapsto by, via \mapsto via]$

VARIABLE *blocks* $\langle\{Tx, \ldots\}, \ldots\rangle$
VARIABLE *next_block* $\{Tx, \ldots\}$
VARIABLE *mempool* $\{Tx, \ldots\}$
VARIABLE *shared_knowledge* $\{Tx, \ldots\}$
VARIABLE *signers_map* $[participant \mapsto \{allowed\_sig, \ldots\}]$
VARIABLE *per_block_enabled* $\langle\{Tx, \ldots\}, \ldots\rangle$

$fullState \triangleq \langle blocks, next\_block, signers\_map, shared\_knowledge, mempool, per\_block\_enabled\rangle$
$unchangedByMM \triangleq \langle blocks, signers\_map, shared\_knowledge, mempool\rangle$

A few generic operators

$Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$
$Min(set) \triangleq \text{CHOOSE } x \in set : \forall y \in set : x \leq y$
$Max(set) \triangleq \text{CHOOSE } x \in set : \forall y \in set : x \geq y$

Various definitions that help to improve readability of the spec

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

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

$tx\_lock\_A \triangleq$ "tx_lock_A"
$tx\_lock\_B \triangleq$ "tx_lock_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"
$tx\_spend\_A \triangleq$ "tx_spend_A"
$tx\_spend\_B \triangleq$ "tx_spend_B"
$tx\_spend\_success \triangleq$ "tx_spend_success"
$tx\_spend\_refund\_1\_alice \triangleq$ "tx_spend_refund_1_alice"
$tx\_spend\_refund\_1\_bob \triangleq$ "tx_spend_refund_1_bob"
$tx\_spend\_revoke \triangleq$ "tx_spend_revoke"
$tx\_spend\_refund\_2 \triangleq$ "tx_spend_refund_2"
$tx\_spend\_timeout \triangleq$ "tx_spend_timeout"

$nLockTime \triangleq$ "nLockTime"
$nSequence \triangleq$ "nSequence"
$NoTimelock \triangleq [days \mapsto 0, type \mapsto nLockTime]$

If blocks per day are low, the absolute locks need to be shifted,
otherwise not all contract paths will be reachable

$$ABS\_LK\_OFFSET \triangleq \text{CASE } BLOCKS\_PER\_DAY = 1 \rightarrow 2 \;\square\; BLOCKS\_PER\_DAY = 2 \rightarrow 1 \;\square\; \text{OTHER} \rightarrow 0$$

The map of the transactions, their possible destinations and timelocks.
Adaptor signatures are modelled by an additional value in the required
signature set – ss . For modelling purposes, the secret acts as just another signature.
ds stands for “destinations”, and lk stands for “lock” (timelocks).
Only blockheight-based timelocks are modelled.

$tx\_map \triangleq [$

’Contract’ transactions – destinations are other transactions

$$\begin{aligned}
tx_lock_A &\mapsto [ds \mapsto \{tx_success, tx_refund_1, tx_revoke, tx_spend_A\},\\
&\quad\; ss \mapsto \{sigAlice\}],\\
tx_lock_B &\mapsto [ds \mapsto \{tx_spend_B\},\\
&\quad\; ss \mapsto \{sigBob\}],\\
tx_success &\mapsto [ds \mapsto \{tx_spend_success\},\\
&\quad\; ss \mapsto \{sigAlice, sigBob, secretBob\}],\\
tx_refund_1 &\mapsto [ds \mapsto \{tx_spend_refund_1_bob, tx_spend_refund_1_alice\},\\
&\quad\; ss \mapsto \{sigAlice, sigBob, secretAlice\},\\
&\quad\; lk \mapsto [days \mapsto ABS_LK_OFFSET + 1, type \mapsto nLockTime]],\\
tx_revoke &\mapsto [ds \mapsto \{tx_refund_2, tx_timeout, tx_spend_revoke\},\\
&\quad\; ss \mapsto \{sigAlice, sigBob\},\\
&\quad\; lk \mapsto [days \mapsto ABS_LK_OFFSET + 2, type \mapsto nLockTime]],\\
tx_refund_2 &\mapsto [ds \mapsto \{tx_spend_refund_2\},\\
&\quad\; ss \mapsto \{sigAlice, sigBob, secretAlice\},\\
&\quad\; lk \mapsto [days \mapsto 1, type \mapsto nSequence]],\\
tx_timeout &\mapsto [ds \mapsto \{tx_spend_timeout\},\\
&\quad\; ss \mapsto \{sigAlice, sigBob\},\\
&\quad\; lk \mapsto [days \mapsto 2, type \mapsto nSequence]],
\end{aligned}$$

'Terminal' transactions – destinations are participants

$$
\begin{array}{ll}
\mathit{tx\_spend\_A} & \mapsto [\mathit{ds} \mapsto \{\mathit{Alice},\ \mathit{Bob}\}, \\
& \quad\ \mathit{ss} \mapsto \{\mathit{sigAlice},\ \mathit{sigBob}\}], \\
\mathit{tx\_spend\_B} & \mapsto [\mathit{ds} \mapsto \{\mathit{Alice},\ \mathit{Bob}\}, \\
& \quad\ \mathit{ss} \mapsto \{\mathit{secretAlice},\ \mathit{secretBob}\}], \\
\mathit{tx\_spend\_success} & \mapsto [\mathit{ds} \mapsto \{\mathit{Bob}\}, \\
& \quad\ \mathit{ss} \mapsto \{\mathit{sigBob}\}], \\
\mathit{tx\_spend\_refund\_1\_bob} & \mapsto [\mathit{ds} \mapsto \{\mathit{Bob}\}, \\
& \quad\ \mathit{ss} \mapsto \{\mathit{sigAlice},\ \mathit{sigBob}\}], \\
\mathit{tx\_spend\_refund\_1\_alice} & \mapsto [\mathit{ds} \mapsto \{\mathit{Alice}\}, \\
& \quad\ \mathit{ss} \mapsto \{\mathit{sigAlice}\}, \\
& \quad\ \mathit{lk} \mapsto [\mathit{days} \mapsto 1,\ \mathit{type} \mapsto \mathit{nSequence}]], \\
\mathit{tx\_spend\_revoke} & \mapsto [\mathit{ds} \mapsto \{\mathit{Alice},\ \mathit{Bob}\}, \\
& \quad\ \mathit{ss} \mapsto \{\mathit{sigAlice},\ \mathit{sigBob}\}], \\
\mathit{tx\_spend\_refund\_2} & \mapsto [\mathit{ds} \mapsto \{\mathit{Alice}\}, \\
& \quad\ \mathit{ss} \mapsto \{\mathit{sigAlice}\}], \\
\mathit{tx\_spend\_timeout} & \mapsto [\mathit{ds} \mapsto \{\mathit{Bob}\}, \\
& \quad\ \mathit{ss} \mapsto \{\mathit{sigBob}\}]
\end{array}
$$

]

$\mathit{all\_transactions} \triangleq \text{DOMAIN } \mathit{tx\_map}$

first_transaction defined so that miner's actions do not need to refer to any contract-specific info, and can just refer to *first_transaction* instead.

$\mathit{first\_transaction} \triangleq \mathit{tx\_lock\_A}$

$\mathit{ConfirmedTransactions} \triangleq \{\mathit{tx.id} : \mathit{tx} \in \text{UNION } \mathit{Range}(\mathit{blocks})\}$

$\mathit{NextBlockTransactions} \triangleq \{\mathit{tx.id} : \mathit{tx} \in \mathit{next\_block}\}$

$\mathit{NextBlockConfirmedTransactions} \triangleq$
$\quad \mathit{ConfirmedTransactions} \cup \mathit{NextBlockTransactions}$

$\mathit{MempoolTransactions} \triangleq \{\mathit{tx.id} : \mathit{tx} \in \mathit{mempool}\}$

$\mathit{SentTransactions} \triangleq \mathit{ConfirmedTransactions} \cup \mathit{MempoolTransactions}$

$EnabledTransactions \triangleq \{tx.id : tx \in \text{UNION } Range(per\_block\_enabled)\}$

$ContractTransactions \triangleq$
$\quad \{id \in all\_transactions :$
$\qquad \forall d \in tx\_map[id].ds : d \in all\_transactions\}$

$TerminalTransactions \triangleq$
$\quad \{id \in all\_transactions :$
$\qquad \forall d \in tx\_map[id].ds : d \in participants\}$

$\text{ASSUME } \forall id \in all\_transactions : \lor id \in TerminalTransactions$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \lor id \in ContractTransactions$

In this contract each transaction has only one parent,
so we can use simple mapping from *dep_id* to parent *id*

$dependency\_map \triangleq$
$\quad [dep\_id \in \text{UNION } \{tx\_map[id].ds : id \in ContractTransactions\}$
$\quad\ \mapsto \text{CHOOSE } id \in ContractTransactions : dep\_id \in tx\_map[id].ds]$

Special destination for the case when funds will still be locked
at the contract after the transaction is spent

$Contract \triangleq$ "Contract"

$DstSet(id) \triangleq$
$\quad \text{IF } id \in ContractTransactions \text{ THEN } \{Contract\} \text{ ELSE } tx\_map[id].ds$

The CASE statement has no 'OTHER' clause - only single dst is expected

$SingleDst(id) \triangleq \text{CASE } id \in ContractTransactions \rightarrow Contract$
$\qquad\qquad\qquad\quad \Box \quad Cardinality(tx\_map[id].ds) = 1$
$\qquad\qquad\qquad\qquad\ \rightarrow \text{CHOOSE } d \in tx\_map[id].ds : \text{TRUE}$

The set of transactions conflicting with the given transaction

$ConflictingSet(id) \triangleq$
$\quad \text{IF } id \in \text{DOMAIN } dependency\_map$
$\quad\ \text{THEN } \{dep\_id \in \text{DOMAIN } dependency\_map :$
$\qquad\qquad dependency\_map[dep\_id] = dependency\_map[id]\}$
$\quad\ \text{ELSE } \{id\}$

Transaction also conflicts with itself

$\text{ASSUME } \forall id \in all\_transactions : id \in ConflictingSet(id)$

$ConfirmationHeight(id) \triangleq$
$\quad \text{CHOOSE } bn \in \text{DOMAIN } blocks : \exists\, tx \in blocks[bn] : tx.id = id$

All the transactions the given transaction depends on.
Because each transaction can only have one dependency in our model,
all dependencies form a chain, not a tree.

$\text{RECURSIVE } DependencyChain(\_)$
$DependencyChain(id) \triangleq$
$\quad \text{IF } id \in \text{DOMAIN } dependency\_map$
$\quad\quad \text{THEN } \{id\} \cup DependencyChain(dependency\_map[id])$
$\quad\quad \text{ELSE } \{id\}$

All the transactions that depend on the given transaction.
Dependants form a tree, but the caller is interested in just a set.

$\text{RECURSIVE } AllDependants(\_)$
$AllDependants(id) \triangleq$
$\quad \text{LET } dependants \triangleq tx\_map[id].ds \setminus participants$
$\quad\quad \text{IN } \quad \text{IF } dependants = \{\}$
$\quad\quad\quad\quad \text{THEN } \{id\}$
$\quad\quad\quad\quad \text{ELSE } dependants \cup \text{UNION } \{AllDependants(d\_id) : d\_id \in dependants\}$

All transactions that cannot ever become valid because other, conflicting
transactions were confirmed befor them

$InvalidatedTransactions \triangleq$
$\quad \text{UNION } \{\{c\_id\} \cup AllDependants(c\_id) \quad : c\_id \in$
$\quad\quad\quad \text{UNION } \{ConflictingSet(id) \setminus \{id\} : id \in ConfirmedTransactions\}\}$

All transactions that is not yet sent/confirmed, and have a chance to be.

$RemainingTransactions \triangleq$
$\quad ((all\_transactions \setminus ConfirmedTransactions) \setminus InvalidatedTransactions)$

$Timelock(id) \triangleq \text{IF } \text{“lk”} \in \text{DOMAIN } tx\_map[id] \text{ THEN } tx\_map[id].lk \text{ ELSE } NoTimelock$

$UnreachableHeight \triangleq 2^{30} + (2^{30} - 1)$

Calculate the height at which the timelock for the given transaction expires, taking $BLOCKS\_PER\_DAY$ and dependencies confirmation into account

$$
\begin{array}{l}
\mathit{TimelockExpirationHeight}(id) \triangleq \\
\quad \text{LET } lk \triangleq \mathit{Timelock}(id) \\
\quad \text{IN} \quad \text{CASE } lk.type = nLockTime \\
\qquad\qquad \rightarrow lk.days * BLOCKS\_PER\_DAY \\
\qquad\quad \Box \quad lk.type = nSequence \\
\qquad\qquad \rightarrow \text{IF } dependency\_map[id] \in \mathit{ConfirmedTransactions} \\
\qquad\qquad\quad \text{THEN } \mathit{ConfirmationHeight}(dependency\_map[id]) \\
\qquad\qquad\qquad + lk.days * BLOCKS\_PER\_DAY \\
\qquad\qquad\quad \text{ELSE } \mathit{UnreachableHeight}
\end{array}
$$

"Hard" deadline for transaction means that it is unsafe to publish the transaction after the deadline

$$
\begin{array}{l}
\mathit{Deadline}(id) \triangleq \\
\quad \text{LET } hs \triangleq \{\mathit{TimelockExpirationHeight}(c\_id) : \\
\qquad\qquad c\_id \in \mathit{ConflictingSet}(id) \setminus \{id\}\} \\
\qquad higher\_hs \triangleq \{h \in hs : h > \mathit{TimelockExpirationHeight}(id)\} \\
\quad \text{IN} \quad \text{IF } higher\_hs = \{\} \\
\qquad \text{THEN } \mathit{UnreachableHeight} \\
\qquad \text{ELSE } \mathit{Min}(higher\_hs)
\end{array}
$$

"Soft" deadline for transaction means that after the deadline, mining the transaction will mean that it was 'stalling' for too long

$$
\begin{array}{l}
\mathit{SoftDeadline}(id) \triangleq \\
\quad \text{LET } dl \triangleq \mathit{Deadline}(id) \\
\qquad h \triangleq \mathit{TimelockExpirationHeight}(id) \\
\quad \text{IN} \quad \text{IF } dl = \mathit{UnreachableHeight} \\
\qquad \text{THEN IF } id \in \mathit{EnabledTransactions} \\
\qquad\qquad \text{THEN } (\text{CHOOSE } en \in \text{DOMAIN } per\_block\_enabled : \\
\qquad\qquad\qquad \exists\, tx \in per\_block\_enabled[en] : tx.id = id) \\
\qquad\qquad\quad + MAX\_DAYS\_STALLING * BLOCKS\_PER\_DAY \\
\qquad\qquad \text{ELSE IF } h \neq \mathit{UnreachableHeight} \\
\qquad\qquad\quad \text{THEN } h + MAX\_DAYS\_STALLING * BLOCKS\_PER\_DAY \\
\qquad\qquad\quad \text{ELSE } 0 \\
\qquad \text{ELSE } dl
\end{array}
$$

$SigsAvailable(id, sender, to) \triangleq$
$\quad \text{LET } secrets\_shared \triangleq$
$\qquad\qquad \text{UNION } \{tx.ss \cap all\_secrets : tx \in shared\_knowledge\}$
$\qquad\quad sigs\_shared \triangleq$
$\qquad\qquad \text{UNION } \{tx.ss : tx \in \{tx \in shared\_knowledge : \land tx.id = id$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land tx.to = to\}\}$
$\quad \text{IN } \quad sigs\_shared \cup secrets\_shared \cup signers\_map[sender]$

$DependencySatisfied(id, ids) \triangleq$
$\quad id \in \text{DOMAIN } dependency\_map \Rightarrow dependency\_map[id] \in ids$

$IsSpendableTx(tx, other\_ids) \triangleq$
$\quad \land \{\} = ConflictingSet(tx.id) \cap other\_ids$
$\quad \land DependencySatisfied(tx.id, other\_ids)$
$\quad \land tx.ss \subseteq SigsAvailable(tx.id, tx.by, tx.to)$
$\quad \land Len(blocks) \geq TimelockExpirationHeight(tx.id)$

Sending *tx_spend_B* does not actually expose secrets, because the secrets are used as keys, and *sigSecretBob* would be exposed rather than *secretBob*. Instead of introducing $revealSecret < Alice \mid Bob >$, $sigSecret < Alice \mid Bob >$ we simply filter out signatures of *tx_spend_B* before placing into shared knowledge

$ShareKnowledge(knowledge) \triangleq$
$\quad \text{LET } knowledge\_filtered \triangleq$
$\qquad\qquad \{\text{IF } tx.id \neq tx\_spend\_B \text{ THEN } tx \text{ ELSE } [tx \text{ EXCEPT } !.ss = \{\}] :$
$\qquad\qquad\; tx \in knowledge\}$

shared_knowledge may not change here, callers need to check if they care

$\quad \text{IN } \quad shared\_knowledge' = shared\_knowledge \cup knowledge\_filtered$

$ShareTransactions(ids, by) \triangleq$
$\quad \text{LET } Ss(id) \triangleq (tx\_map[id].ss \cap signers\_map[by]) \setminus all\_secrets$
$\qquad\quad txs \triangleq \{Tx(id, Ss(id), by, SingleDst(id), \text{“direct”}) : id \in ids\}$
$\quad \text{IN } \quad \land ShareKnowledge(txs)$
$\qquad\quad \land shared\_knowledge' \neq shared\_knowledge$ not a new *knowledge* $\Rightarrow$ fail

Txs enabled at the current cycle, used to update *per_block_enabled* vector

$NewlyEnabledTxs \triangleq$

$\{tx \in$
$\quad \text{UNION}$
$\quad \{\text{UNION}$
$\quad\quad \{$
$\quad\quad\quad \{$
$\quad\quad\quad\quad Tx(id, tx\_map[id].ss, sender, to, \text{“enabled”}) : to \in DstSet(id)$
$\quad\quad\quad \} : id \in RemainingTransactions$
$\quad\quad \} : sender \in participants$
$\quad \} : \land \neg\exists\, etx \in \text{UNION } Range(per\_block\_enabled) : etx.id = tx.id$
$\quad\quad \land IsSpendableTx(tx, ConfirmedTransactions)$
$\}$

$SendTransactionToMempool(id, sender, to) \triangleq$

$\text{LET } tx \triangleq Tx(id, tx\_map[id].ss, sender, to, \text{“mempool”})$
$\text{IN } \land IsSpendableTx(tx, SentTransactions)$
$\quad \land Len(blocks) < Deadline(id)$
$\quad \land mempool' = mempool \cup \{tx\}$
$\quad \land ShareKnowledge(\{tx\})$

Give *tx* directly to miner, bypassing global *mempool*
No *Deadline* check because information is not shared,
and after the block is mined, there's no possible contention
unless the block is orphaned. Orphan blocks are not modelled,
and therefore there's no need for additional restriction
as any state space restriction can possibly mask some other issue

$SendTransactionToMiner(id, sender, to) \triangleq$

$\land STEALTHY\_SEND\_POSSIBLE$
$\land \text{LET } tx \triangleq Tx(id, tx\_map[id].ss, sender, to, \text{“miner”})$
$\quad \text{IN } \land IsSpendableTx(tx, NextBlockConfirmedTransactions)$
$\quad\quad \land next\_block' = next\_block \cup \{tx\}$

$$
\begin{array}{l}
\mathit{SendTransaction}(id, sender, to) \triangleq \\
\quad \lor \land \mathit{SendTransactionToMempool}(id, sender, to) \\
\quad \land \textsc{unchanged}\ next\_block \\
\quad \lor \land \mathit{SendTransactionToMiner}(id, sender, to) \\
\quad \land \textsc{unchanged}\ \langle mempool, shared\_knowledge \rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{SendSomeTransaction}(ids, sender) \triangleq \\
\quad \textsc{let}\ \mathit{SendSome}(filtered\_ids) \triangleq \\
\qquad\qquad \exists\, id \in filtered\_ids : \\
\qquad\qquad \exists\, to \in (\textsc{if}\ id \in \mathit{ContractTransactions} \\
\qquad\qquad\qquad\qquad \textsc{then}\ \{\mathit{Contract}\} \\
\qquad\qquad\qquad\qquad \textsc{else}\ \ tx\_map[id].ds \cap \{sender\}) : \\
\qquad\qquad\quad \mathit{SendTransaction}(id, sender, to) \\
\qquad\quad terminal\_ids \triangleq ids \cap \mathit{TerminalTransactions} \\
\quad \textsc{in}\quad \textsc{case}\ \mathit{PARTICIPANTS\_IRRATIONAL} \\
\qquad\qquad\quad \rightarrow \mathit{SendSome}(ids) \quad \text{Irrational participants do no prioritization} \\
\qquad\qquad \Box \quad \textsc{enabled}\ \mathit{SendSome}(terminal\_ids) \\
\qquad\qquad\quad \rightarrow \mathit{SendSome}(terminal\_ids) \quad \text{Can send terminal } tx \Rightarrow \text{ do it immediately} \\
\qquad\qquad \Box \quad \textsc{other} \\
\qquad\qquad\quad \rightarrow \mathit{SendSome}(ids \setminus terminal\_ids)
\end{array}
$$

$$
\begin{array}{l}
\mathit{HasCustody}(ids, participant) \triangleq \\
\quad \exists\, id \in ids : \exists\, tx \in \textsc{union}\ \mathit{Range}(blocks) : tx.id = id \land tx.to = participant
\end{array}
$$

Sharing secrets or keys has to occur before deadline to send $tx\_success$

$$
\mathit{TooLateToShare} \triangleq \mathit{Len}(blocks) \geq \mathit{Deadline}(tx\_success)
$$

Participant actions

Transactions *Alice* initially shares signatures on

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

Transactions *Bob* initially shares signatures on

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

Conditions to divide the contract execution into phases according to original spec

$$
\begin{aligned}
&Phase\_3\_cond \triangleq tx\_lock\_B \in ConfirmedTransactions \\
&Phase\_2\_cond \triangleq tx\_lock\_A \in ConfirmedTransactions \\
&Phase\_1\_cond \triangleq \\
&\quad \land \forall id \in phase0\_to\_share\_Alice : \\
&\qquad \exists tx \in shared\_knowledge : tx.id = id \land sigAlice \in tx.ss \\
&\quad \land \forall id \in phase0\_to\_share\_Bob : \\
&\qquad \exists tx \in shared\_knowledge : tx.id = id \land sigBob \in tx.ss
\end{aligned}
$$

$$
\begin{aligned}
&InPhase\_3 \triangleq \\
&\quad \land Phase\_3\_cond
\end{aligned}
$$

$$
\begin{aligned}
&InPhase\_2 \triangleq \\
&\quad \land Phase\_2\_cond \\
&\quad \land \neg Phase\_3\_cond
\end{aligned}
$$

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

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

Helper operators to declutter the action expressions

$$
\begin{aligned}
&NoSending \triangleq \text{UNCHANGED } \langle mempool, next\_block \rangle \\
&NoKeysShared \triangleq \text{UNCHANGED } signers\_map \\
&NoKnowledgeShared \triangleq \text{UNCHANGED } shared\_knowledge
\end{aligned}
$$

```
AliceAction ≜
    LET Send(ids) ≜ SendSomeTransaction(ids, Alice)
        Share(ids) ≜ ShareTransactions(ids, Alice)
         SafeToSend(id) ≜
             CASE PARTICIPANTS_IRRATIONAL
                    → TRUE  Unsafe txs are OK for irrational Alice
               □  id = tx_refund_1  Do not send refund_1 if tx_success was shared
                    → tx_success ∉ {tx.id : tx ∈ shared_knowledge}
               □  secretAlice ∈ tx_map[id].ss
                     Once Alice received secretBob, should never send out secretAlice
                    → ∨ secretBob ∉ signers_map[Alice]
                       ∨ id = tx_spend_B  unless this is a transaction to get B
                                          which does not in fact expose secrets
               □  OTHER  → TRUE
    IN    ∨ ∧ InPhase_0
            ∧ Share(phase0_to_share_Alice)
            ∧ NoSending ∧ NoKeysShared
          ∨ ∧ InPhase_1
            ∧ Send({tx_lock_A})
            ∧ NoKeysShared
          ∨ ∧ InPhase_2  Just waiting for Bob to lock B
            ∧ Send({id ∈ RemainingTransactions : SafeToSend(id)})
            ∧ NoKeysShared
          ∨ ∧ InPhase_3
            ∧ ∨ ∧ secretBob ∈ signers_map[Alice]  Bob gave Alice his secret
                ∧ sigAlice ∉ signers_map[Bob]      Alice did not yet gave Bob her key
                ∧ ¬TooLateToShare
                ∧ signers_map′ = [signers_map   Give Alice's key to Bob
                                    EXCEPT ![Bob] = @ ∪ {sigAlice}]
                ∧ NoSending ∧ NoKnowledgeShared
              ∨ ∧ tx_refund_1 ∉ SentTransactions
                ∧ ¬TooLateToShare
                ∧ Share({tx_success})  refund_1 not sent yet, can share
                ∧ NoSending ∧ NoKeysShared
              ∨ ∧ Send({id ∈ RemainingTransactions : SafeToSend(id)})
                ∧ NoKeysShared
```

$$
\begin{array}{l}
BobAction \triangleq \\
\quad \textsc{let}\ Send(ids) \triangleq SendSomeTransaction(ids, Bob) \\
\qquad\quad Share(ids) \triangleq ShareTransactions(ids, Bob) \\
\qquad\quad tx\_success\_sigs \triangleq SigsAvailable(tx\_success, Bob, Contract) \\
\quad \textsc{in}\quad \lor \land InPhase\_0 \\
\qquad\qquad\quad \land Share(phase0\_to\_share\_Bob) \\
\qquad\qquad\quad \land NoSending \land NoKeysShared \\
\qquad\qquad \lor \land InPhase\_1 \quad \text{Just waiting for } Alice \text{ to lock A} \\
\qquad\qquad\quad \land NoSending \land NoKnowledgeShared \land NoKeysShared \\
\qquad\qquad \lor \land InPhase\_2 \\
\qquad\qquad\quad \land Send(\{tx\_lock\_B\}) \\
\qquad\qquad\quad \land NoKeysShared \\
\qquad\qquad \lor \land InPhase\_3 \\
\qquad\qquad\quad \land \lor \land sigAlice \in tx\_success\_sigs \\
\qquad\qquad\qquad\qquad \text{If } Bob \text{ already knows } secretAlice\text{, he doesn't need to share } secretBob \\
\qquad\qquad\qquad\quad \land secretAlice \notin tx\_success\_sigs \\
\qquad\qquad\qquad\quad \land secretBob \notin signers\_map[Alice] \\
\qquad\qquad\qquad\quad \land \neg TooLateToShare \\
\qquad\qquad\qquad\quad \land signers\_map' = [signers\_map \quad \text{Give } secretBob \text{ to } Alice \\
\qquad\qquad\qquad\qquad\qquad\qquad \textsc{except}\ ![Alice] = @ \cup \{secretBob\}] \\
\qquad\qquad\qquad\quad \land NoSending \land NoKnowledgeShared \\
\qquad\qquad\qquad \lor \land Send(RemainingTransactions) \\
\qquad\qquad\qquad\quad \land NoKeysShared
\end{array}
$$

$$MempoolMonitorActionRequired \triangleq$$
$$\exists\, tx \in mempool : \land\ Len(blocks) + 1 = Deadline(tx.id)$$
$$\qquad\qquad\qquad\quad \land\ tx.id \notin NextBlockTransactions$$

We update *next_block* directly rather than having to deal with fees and prioritization. What we want to model is the behavior of participants where once they have sent the transaction, they do anything possible to meet the deadline set by the protocol to confirm the transaction. Failure to do so before the deadline is out of scope, even though it could be caused by some unexpected *mempool* behavior.

Exact *mempool* behavior is too low-level and is better modelled separately to check that high-level constraints can be met. Although if we were to have more complex model where the amounts available for each participant are tracked, it might make sense to include the fees and *mempool* behavior into the model of the contract to catch the cases when participants just can't bump fees anymore, for example.

We could just not model the *mempool* monitoring, and constrain state space such that states with late *txs* are invalid, to express that we don't care about the cases when participants fail to get their *txs* confirmed in time. But maybe there could be some interesting behaviors to be modelled if more elaborate monitor action is implemented

$$MempoolMonitorAction \triangleq$$
$$\text{LET } tx \triangleq \text{CHOOSE } tx \in mempool : Len(blocks) + 1 = Deadline(tx.id)$$
$$\quad txs\_to\_bump \triangleq \{tx\} \cup \{dptx \in mempool :$$
$$\qquad\qquad \land\ tx.id \in \text{DOMAIN } dependency\_map$$
$$\qquad\qquad \land\ dptx.id = dependency\_map[tx.id]$$
$$\qquad\qquad \land\ dptx.id \notin NextBlockTransactions\}$$
$$\text{IN} \quad next\_block' =$$
$$\quad \{nbtx \in next\_block : \text{ conflicting } txs \text{ are expunged from } next\_block$$
$$\qquad \{\} = DependencyChain(nbtx.id) \cap$$
$$\qquad\qquad \text{UNION } \{ConflictingSet(bmptx.id) : bmptx \in txs\_to\_bump\}\}$$
$$\quad \cup \{[bmptx \text{ EXCEPT } !.via = \text{“fee-bump”}] : bmptx \in txs\_to\_bump\}$$

Miner action

$$
\begin{array}{l}
\mathit{IncludeTxIntoBlock} \triangleq \\
\quad \land \exists\, tx \in \mathit{mempool} : \\
\qquad \land \{\} = \mathit{ConflictingSet}(tx.id) \cap \mathit{NextBlockConfirmedTransactions} \\
\qquad \land \mathit{DependencySatisfied}(tx.id,\ \mathit{NextBlockConfirmedTransactions}) \\
\qquad \land \mathit{next\_block}' = \mathit{next\_block} \cup \{tx\} \\
\quad \land \textsc{unchanged}\ \langle \mathit{blocks},\ \mathit{mempool},\ \mathit{shared\_knowledge} \rangle
\end{array}
$$

Needed to restrict the state space, so that model checking is feasible

$$
\begin{array}{l}
\mathit{CanMineEmptyBlock} \triangleq \\
\quad \land \mathit{first\_transaction} \in \mathit{ConfirmedTransactions} \\
\quad \land \textsc{let}\ \mathit{soft\_dls} \triangleq \{\mathit{SoftDeadline}(id) : id \in \mathit{RemainingTransactions}\} \\
\qquad \textsc{in}\ \ \mathit{soft\_dls} \neq \{\} \land \mathit{Len}(\mathit{blocks}) + 1 < \mathit{Max}(\mathit{soft\_dls})
\end{array}
$$

$$
\begin{array}{l}
\mathit{MineTheBlock} \triangleq \\
\quad \textsc{if}\ \mathit{next\_block} = \{\} \\
\quad\ \textsc{then}\ \land \mathit{CanMineEmptyBlock} \\
\qquad\qquad \land \mathit{blocks}' = \mathit{Append}(\mathit{blocks},\ \{\}) \\
\qquad\qquad \land \textsc{unchanged}\ \langle \mathit{mempool},\ \mathit{next\_block},\ \mathit{shared\_knowledge} \rangle \\
\quad\ \textsc{else}\ \ \land \mathit{blocks}' = \mathit{Append}(\mathit{blocks},\ \mathit{next\_block}) \\
\qquad\qquad \land \mathit{mempool}' = \\
\qquad\qquad\quad \{tx \in \mathit{mempool} : \ \text{conflicting } \mathit{txs} \text{ are expunged from } \mathit{mempool} \\
\qquad\qquad\quad\ \{\} = \mathit{DependencyChain}(tx.id) \cap \\
\qquad\qquad\qquad\quad \textsc{union}\ \{\mathit{ConflictingSet}(nbtx.id) : nbtx \in \mathit{next\_block}\}\} \\
\qquad\qquad \land \mathit{next\_block}' = \{\} \\
\qquad\qquad \land \mathit{ShareKnowledge}(\mathit{next\_block} \setminus \mathit{mempool})
\end{array}
$$

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

Auxiliary action for soft-deadline tracking

$$
\begin{array}{l}
\mathit{UpdateEnabledPerBlock} \triangleq \\
\quad \mathit{per\_block\_enabled}' = \\
\qquad \textsc{if}\ \mathit{Len}(\mathit{per\_block\_enabled}) < \mathit{Len}(\mathit{blocks}) + 1 \\
\qquad\ \textsc{then}\ \mathit{Append}(\mathit{per\_block\_enabled},\ \mathit{NewlyEnabledTxs}) \\
\qquad\ \textsc{else}\ \ [\mathit{per\_block\_enabled}\ \textsc{except}\ ![\mathit{Len}(\mathit{blocks}) + 1] = @ \cup \mathit{NewlyEnabledTxs}]
\end{array}
$$

High-level contract spec

First, the 'unnatural' cases.

For all transactions defined by the original spec
to be covered by the model, we need to also model the case
where *Alice* misbehaves by sending transactions containing her
secret after she gave *tx_success to Bob.This behavior
also enables Bob to misbehave by failing to punish Alice s*
misbehavior, which results in *Bob* losing *B*.
The following four actions are needed to express all that.

$$
\begin{array}{l}
AliceLostByMisbehaving \triangleq \\
\quad \land\ HasCustody(\{tx\_spend\_B\},\ Bob) \\
\quad \land\ HasCustody(\{tx\_spend\_refund\_1\_bob\},\ Bob)
\end{array}
$$

$$
\begin{array}{l}
BobLostByBeingLateOnRefund\_1 \triangleq \\
\quad \land\ HasCustody(\{tx\_spend\_B\},\ Alice) \\
\quad \land\ HasCustody(\{tx\_spend\_refund\_1\_alice\},\ Alice)
\end{array}
$$

$$
\begin{array}{l}
BobLostByBeingLateOnRefund\_2 \triangleq \\
\quad \land\ HasCustody(\{tx\_spend\_B\},\ Alice) \\
\quad \land\ HasCustody(\{tx\_spend\_refund\_2\},\ Alice)
\end{array}
$$

$$
\begin{array}{l}
SwapUnnaturalEnding \triangleq \\
\quad \lor\ AliceLostByMisbehaving \\
\quad \lor\ BobLostByBeingLateOnRefund\_1 \\
\quad \lor\ BobLostByBeingLateOnRefund\_2
\end{array}
$$

The normal, 'natural' cases.

$$
\begin{array}{l}
\mathit{SwapSuccessful} \triangleq \\
\quad \land\ \mathit{HasCustody}(\{\mathit{tx\_spend\_B}\},\ \mathit{Alice}) \\
\quad \land\ \lor\ \mathit{HasCustody}(\{\mathit{tx\_spend\_A},\ \mathit{tx\_spend\_success}, \\
\qquad\qquad\qquad\qquad \mathit{tx\_spend\_timeout},\ \mathit{tx\_spend\_revoke}\},\ \mathit{Bob}) \\
\qquad \lor\ \land\ \mathit{PARTICIPANTS\_IRRATIONAL} \\
\qquad\quad \land\ \mathit{HasCustody}(\{\mathit{tx\_spend\_refund\_1\_bob}\},\ \mathit{Bob})
\end{array}
$$

$$
\begin{array}{l}
\mathit{SwapAborted} \triangleq \\
\quad \land\ \mathit{HasCustody}(\{\mathit{tx\_spend\_A},\ \mathit{tx\_spend\_refund\_1\_alice},\ \mathit{tx\_spend\_refund\_2}\},\ \mathit{Alice}) \\
\quad \land\ \lor\ \mathit{HasCustody}(\{\mathit{tx\_spend\_B}\},\ \mathit{Bob}) \\
\qquad \lor\ \mathit{tx\_lock\_B} \notin \mathit{SentTransactions}
\end{array}
$$

$$
\begin{array}{l}
\mathit{SwapTimedOut} \triangleq \\
\quad \land\ \mathit{tx\_spend\_timeout} \in \mathit{ConfirmedTransactions}
\end{array}
$$

Alice can't claim *tx_spend_B* on timeout

$$
\begin{array}{l}
\quad \land\ \mathit{secretBob} \notin \mathit{signers\_map}[\mathit{Alice}] \\
\quad \land\ \mathit{secretBob} \notin \textsc{union}\ \{\mathit{tx.ss} : \mathit{tx} \in \mathit{shared\_knowledge}\}
\end{array}
$$

All possible endings of the contract

$$
\begin{array}{ll}
\mathit{ContractFinished} \triangleq & \lor\ \mathit{SwapSuccessful} \\
& \lor\ \mathit{SwapAborted} \\
& \lor\ \mathit{SwapTimedOut} \\
& \lor\ \mathit{PARTICIPANTS\_IRRATIONAL} \land \mathit{SwapUnnaturalEnding}
\end{array}
$$

Actions in the contract when it is not yet finished. Separated into dedicated operator to be able to test ENABLED *ContractAction*

$$
\begin{array}{l}
\mathit{ContractAction} \triangleq \\
\quad \lor\ \mathit{AliceAction} \qquad\qquad\quad \land\ \textsc{unchanged}\ \mathit{blocks} \\
\quad \lor\ \mathit{BobAction} \qquad\qquad\quad\ \land\ \textsc{unchanged}\ \mathit{blocks} \\
\quad \lor\ \textsc{if}\ \mathit{MempoolMonitorActionRequired} \\
\qquad \textsc{then}\ \mathit{MempoolMonitorAction} \land \textsc{unchanged}\ \mathit{unchangedByMM} \\
\qquad \textsc{else}\ \ \mathit{MinerAction} \qquad\qquad\quad \land\ \textsc{unchanged}\ \mathit{signers\_map}
\end{array}
$$

Invariants

$$
\begin{array}{l}
TypeOK \triangleq \\
\quad \text{LET } TxConsistent(tx, vias) \triangleq \land tx.id \in all\_transactions \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land tx.ss \subseteq tx\_map[tx.id].ss \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land tx.to \in DstSet(tx.id) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land tx.by \in participants \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land tx.via \in vias \\
\qquad\quad AllSigsPresent(tx) \triangleq tx.ss = tx\_map[tx.id].ss \\
\qquad\quad SigConsistent(sig) \triangleq \land sig.id \in all\_transactions \\
\qquad\qquad\qquad\qquad\qquad\qquad \land sig.s \in all\_sigs \\
\qquad\qquad\qquad\qquad\qquad\qquad \land sig.ds \subseteq participants \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \cup \text{ DOMAIN } dependency\_map \\
\quad \text{IN} \quad \land \forall tx \in \text{UNION } Range(blocks) : \\
\qquad\qquad \lor \land TxConsistent(tx, \{\text{"mempool"}, \text{"miner"}, \text{"fee-bump"}\}) \\
\qquad\qquad\quad \land AllSigsPresent(tx) \\
\qquad\qquad \lor Print(\langle \text{"\sim TypeOK blocks"}, tx\rangle, \text{FALSE}) \\
\qquad\quad \land \forall tx \in \text{UNION } Range(per\_block\_enabled) : \\
\qquad\qquad \lor \land TxConsistent(tx, \{\text{"enabled"}\}) \\
\qquad\qquad\quad \land AllSigsPresent(tx) \\
\qquad\qquad \lor Print(\langle \text{"\sim TypeOK blocks"}, tx\rangle, \text{FALSE}) \\
\qquad\quad \land \forall tx \in next\_block : \\
\qquad\qquad \lor \land TxConsistent(tx, \{\text{"mempool"}, \text{"miner"}, \text{"fee-bump"}\}) \\
\qquad\qquad\quad \land AllSigsPresent(tx) \\
\qquad\qquad \lor Print(\langle \text{"\sim TypeOK next\_block"}, tx\rangle, \text{FALSE}) \\
\qquad\quad \land \forall tx \in mempool : \\
\qquad\qquad \lor \land TxConsistent(tx, \{\text{"mempool"}\}) \\
\qquad\qquad\quad \land AllSigsPresent(tx) \\
\qquad\qquad \lor Print(\langle \text{"\sim TypeOK mempool"}, tx\rangle, \text{FALSE}) \\
\qquad\quad \land \forall tx \in shared\_knowledge : \\
\qquad\qquad \lor TxConsistent(tx, \{\text{"mempool"}, \text{"miner"}, \text{"fee-bump"}, \text{"direct"}\}) \\
\qquad\qquad \lor Print(\langle \text{"\sim TypeOK shared\_knowledge"}, tx\rangle, \text{FALSE}) \\
\qquad\quad \land \forall p \in \text{DOMAIN } signers\_map : \\
\qquad\qquad \lor p \in participants \land \forall sig \in signers\_map[p] : sig \in all\_sigs \\
\qquad\qquad \lor Print(\langle \text{"\sim TypeOK signers\_map"}, p\rangle, \text{FALSE})
\end{array}
$$

$$
\begin{array}{l}
\mathit{ConsistentPhase} \triangleq \\
\quad \text{LET } \mathit{phases} \triangleq \langle \mathit{InPhase\_0}, \mathit{InPhase\_1}, \mathit{InPhase\_2}, \mathit{InPhase\_3} \rangle \\
\quad \text{IN } \mathit{Cardinality}(\{i \in \text{DOMAIN } \mathit{phases} : \mathit{phases}[i]\}) = 1
\end{array}
$$

$$
\begin{array}{l}
\mathit{OnlyWhenParticipantsAreRational} \triangleq \\
\quad \mathit{PARTICIPANTS\_IRRATIONAL} \\
\qquad \Rightarrow \mathit{Assert}(\text{FALSE}, \text{"Not applicable when participants are not rational"})
\end{array}
$$

$$
\begin{array}{l}
\mathit{NoConcurrentSecretKnowledge} \triangleq \\
\quad \land\ \mathit{OnlyWhenParticipantsAreRational} \\
\quad \land\ \text{LET } \mathit{SecretsShared} \triangleq \\
\qquad\qquad (\mathit{all\_secrets} \cap \text{UNION } \{\mathit{tx.ss} : \mathit{tx} \in \mathit{shared\_knowledge}\}) \\
\qquad\qquad \cup (\{\mathit{secretBob}\} \cap \mathit{signers\_map}[\mathit{Alice}]) \\
\qquad\qquad \cup (\{\mathit{secretAlice}\} \cap \mathit{signers\_map}[\mathit{Bob}]) \\
\qquad \text{IN } \mathit{Cardinality}(\mathit{SecretsShared}) \leq 1
\end{array}
$$

$$
\begin{array}{l}
\mathit{NoUnexpectedTransactions} \triangleq \\
\quad \land\ \mathit{OnlyWhenParticipantsAreRational} \\
\quad \land\ \mathit{tx\_spend\_refund\_1\_bob} \notin \mathit{SentTransactions}
\end{array}
$$

$$
\begin{array}{l}
\mathit{NoConflictingTransactions} \triangleq \\
\quad \text{LET } \mathit{ConflictCheck}(\mathit{txs}) \triangleq \\
\qquad\qquad \text{LET } \mathit{ids} \triangleq \{\mathit{tx.id} : \mathit{tx} \in \mathit{txs}\} \\
\qquad\qquad \text{IN } \land\ \mathit{Cardinality}(\mathit{ids}) = \mathit{Cardinality}(\mathit{txs}) \\
\qquad\qquad\qquad \land\ \forall\, \mathit{id} \in \mathit{ids} : \mathit{ConflictingSet}(\mathit{id}) \cap \mathit{ids} = \{\mathit{id}\} \\
\quad \text{IN } \land\ \mathit{ConflictCheck}(\text{UNION } \mathit{Range}(\mathit{blocks}) \cup \mathit{next\_block}) \\
\qquad\quad \land\ \mathit{ConflictCheck}(\text{UNION } \mathit{Range}(\mathit{blocks}) \cup \mathit{mempool})
\end{array}
$$

$$
\begin{array}{l}
\mathit{NoSingleParticipantTakesAll} \triangleq \\
\quad \land\ \mathit{OnlyWhenParticipantsAreRational} \\
\quad \land\ \forall\, p \in \mathit{participants} : \\
\qquad \text{LET } \mathit{txs\_to\_p} \triangleq \{\mathit{tx} \in \text{UNION } \mathit{Range}(\mathit{blocks}) : \mathit{tx.to} = p\} \\
\qquad \text{IN } \mathit{Cardinality}(\{\mathit{tx.id} : \mathit{tx} \in \mathit{txs\_to\_p}\}) \leq 1
\end{array}
$$

$$
\begin{array}{l}
\mathit{TransactionTimelocksEnforced} \triangleq \\
\quad \land\ \forall\, \mathit{tx} \in \mathit{mempool} : \mathit{Len}(\mathit{blocks}) \geq \mathit{TimelockExpirationHeight}(\mathit{tx.id}) \\
\quad \land\ \mathit{STEALTHY\_SEND\_POSSIBLE} \\
\qquad \Rightarrow \forall\, \mathit{tx} \in \mathit{next\_block} : \mathit{Len}(\mathit{blocks}) \geq \mathit{TimelockExpirationHeight}(\mathit{tx.id})
\end{array}
$$

$$
\begin{array}{l}
\mathit{ExpectedStateOnAbortOrTimeout} \triangleq \\
\quad \mathit{SwapAborted} \lor \mathit{SwapTimedOut} \\
\quad \Rightarrow \textsc{let}\ \mathit{ids\_left} \triangleq \textsc{if enabled}\ \mathit{ContractAction}\ \textsc{then}\ \{\mathit{tx\_lock\_B}\}\ \textsc{else}\ \{\} \\
\qquad \textsc{in}\quad \mathit{RemainingTransactions} \subseteq \{\mathit{tx\_spend\_B}\} \cup \mathit{ids\_left}
\end{array}
$$

$$
\begin{array}{l}
\mathit{ExpectedStateOnSuccess} \triangleq \\
\quad \mathit{SwapSuccessful} \Rightarrow \begin{array}[t]{l} \land \lnot \textsc{enabled}\ \mathit{ContractAction} \\ \land \mathit{RemainingTransactions} = \{\} \\ \land \mathit{mempool} = \{\} \\ \land \mathit{next\_block} = \{\} \end{array}
\end{array}
$$

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

$$\mathit{CounterExample} \triangleq \textsc{true} \land \ldots$$

Temporal properties

$$\mathit{ContractEventuallyFinished} \triangleq \Diamond \mathit{ContractFinished}$$

Init & Next

$$
\begin{array}{l}
\mathit{Init} \triangleq \\
\quad \land\ \mathit{blocks} = \langle\rangle \\
\quad \land\ \mathit{per\_block\_enabled} = \langle\rangle \\
\quad \land\ \mathit{next\_block} = \{\} \\
\quad \land\ \mathit{mempool} = \{\} \\
\quad \land\ \mathit{shared\_knowledge} = \{\} \\
\quad \land\ \mathit{signers\_map} = [\mathit{Alice} \mapsto \{\mathit{sigAlice}, \mathit{secretAlice}\}, \\
\qquad\qquad\qquad\qquad\ \ \mathit{Bob} \ \ \mapsto \{\mathit{sigBob}, \mathit{secretBob}\}]
\end{array}
$$

$$
\begin{array}{l}
\mathit{Next} \triangleq \begin{array}[t]{l} \lor \land\ \mathit{ContractAction} \\ \quad \land\ \mathit{UpdateEnabledPerBlock} \\ \lor\ \mathit{ContractFinished} \land \textsc{unchanged}\ \mathit{fullState} \end{array}
\end{array}
$$

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