MODULE *SASwap*

SASwap TLA+ specification (c) by Dmitry Petukhov (https://github.com/dgpv)

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

CONSTANT $BLOCKS\_PER\_DAY$

CONSTANT $STEALTHY\_SEND\_POSSIBLE$
ASSUME $STEALTHY\_SEND\_POSSIBLE \in$ BOOLEAN

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

$Tx(id, ss, ds, by, to, via) \triangleq$
$\quad [id \mapsto id, ss \mapsto ss, ds \mapsto ds, to \mapsto to, by \mapsto by, via \mapsto via]$

VARIABLE *blocks* $\langle\{Tx, \ldots\}, \ldots\rangle$
VARIABLE $next\_block$ $\{Tx, \ldots\}$
VARIABLE *mempool* $\{Tx, \ldots\}$
VARIABLE $shared\_knowledge$ $\{Tx, \ldots\}$
VARIABLE $signers\_map$ $[participant \mapsto \{allowed\_sig, \ldots\}]$

$excluded\_transactions$ is used to track which transactions cannot be confirmed anymore because other, conflicting transactions are already mined. Without this variable, *ContractIsLate* operator will be more complex and possibly slower
VARIABLE $excluded\_transactions$ $\{id, \ldots\}$

$blockState \triangleq \langle blocks, excluded\_transactions\rangle$
$fullState \triangleq \langle blockState, next\_block, signers\_map, shared\_knowledge, mempool\rangle$
$allExceptNextBlock \triangleq \langle blockState, signers\_map, shared\_knowledge, mempool\rangle$

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

$tx\_start\_A \triangleq$ "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"
$tx\_spend\_A \triangleq$ "tx_spend_A"
$tx\_spend\_B \triangleq$ "tx_spend_B"
$tx\_spend\_success \triangleq$ "tx_spend_success"
$tx\_spend\_refund\_1 \triangleq$ "tx_spend_refund_1"
$tx\_spend\_revoke \triangleq$ "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"
$NoTimelock \triangleq [days \mapsto 0, type \mapsto nLockTime]$

The map of the transactions, their possible destinations and timelocks.

Adaptor signatures are modelled by an additional value in the required signature set. For modelling purposes, the secret acts as just another *sig*.

Note: *ds* stands for "destinations", *ss* for "signatures"

$tx\_map \triangleq [$

'Contract' transactions – destinations are other transactions

$$\begin{array}{lll}
tx_start_A & \mapsto \{[& ds \mapsto \{tx_success, tx_refund_1, tx_revoke, tx_spend_A\}, \\
 & & ss \mapsto \{sigAlice\}]\}, \\
tx_start_B & \mapsto \{[& ds \mapsto \{tx_spend_B\}, \\
 & & ss \mapsto \{sigBob\}]\}, \\
tx_success & \mapsto \{[& ds \mapsto \{tx_spend_success\}, \\
 & & ss \mapsto \{sigAlice, sigBob, secretBob\}]\}, \\
tx_refund_1 & \mapsto \{[& ds \mapsto \{tx_spend_refund_1\}, \\
 & & ss \mapsto \{sigAlice, sigBob, secretAlice\}, \\
 & & lk \mapsto [days \mapsto 1, type \mapsto nLockTime]]\}, \\
tx_revoke & \mapsto \{[& ds \mapsto \{tx_refund_2, tx_timeout, tx_spend_revoke\}, \\
 & & ss \mapsto \{sigAlice, sigBob\}, \\
 & & lk \mapsto [days \mapsto 2, type \mapsto nLockTime]]\}, \\
tx_refund_2 & \mapsto \{[& ds \mapsto \{tx_spend_refund_2\}, \\
 & & ss \mapsto \{sigAlice, sigBob, secretAlice\}, \\
 & & lk \mapsto [days \mapsto 1, type \mapsto nSequence]]\}, \\
tx_timeout & \mapsto \{[& ds \mapsto \{tx_spend_timeout\}, \\
 & & ss \mapsto \{sigAlice, sigBob\}, \\
 & & lk \mapsto [days \mapsto 2, type \mapsto nSequence]]\},
\end{array}$$

'Terminal' transactions – destinations are participants

$$\begin{aligned}
&tx_spend_A && \mapsto \{[ds \mapsto \{Alice,\ Bob\}, \\
& && \quad\ ss \mapsto \{sigAlice,\ sigBob\}]\}, \\
&tx_spend_B && \mapsto \{[ds \mapsto \{Alice,\ Bob\}, \\
& && \quad\ ss \mapsto \{secretAlice,\ secretBob\}]\}, \\
&tx_spend_success && \mapsto \{[ds \mapsto \{Bob\}, \\
& && \quad\ ss \mapsto \{sigBob\}]\}, \\
&tx_spend_refund_1 && \mapsto \{[ds \mapsto \{Alice,\ Bob\}, \\
& && \quad\ ss \mapsto \{sigAlice,\ sigBob\}], \\
& && \quad [ds \mapsto \{Alice\}, \\
& && \quad\ ss \mapsto \{sigAlice\}, \\
& && \quad\ lk \mapsto [days \mapsto 1,\ type \mapsto nSequence]]\}, \\
&tx_spend_revoke && \mapsto \{[ds \mapsto \{Alice,\ Bob\}, \\
& && \quad\ ss \mapsto \{sigAlice,\ sigBob\}]\}, \\
&tx_spend_refund_2 && \mapsto \{[ds \mapsto \{Alice\}, \\
& && \quad\ ss \mapsto \{sigAlice\}]\}, \\
&tx_spend_timeout && \mapsto \{[ds \mapsto \{Bob\}, \\
& && \quad\ ss \mapsto \{sigBob\}]\}
\end{aligned}$$

]

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

No variants for transaction with identical destination sets are allowed, because we use $(id,\ ds)$ to identify a transaction variant

$$\text{ASSUME } \forall\, vset \in Range(tx\_map) : Cardinality(\{v[\text{"ds"}] : v \in vset\}) = Cardinality(vset)$$

Will fail if there's more than one variant

$$SoleVariant(id) \triangleq \text{CHOOSE } v \in tx\_map[id] : Cardinality(tx\_map[id]) = 1$$

$$\text{ASSUME } BLOCKS\_PER\_DAY \geq \text{IF } SoleVariant(tx\_refund\_1).lk.days > 1 \text{ THEN } 2 \text{ ELSE } 3$$

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

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

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

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

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

$$\begin{aligned}\mathit{ContractTransactions} \triangleq\ & \{id \in all\_transactions : \\ & \quad \forall\, variant \in tx\_map[id] : \\ & \quad \forall\, d \in variant.ds : d \in all\_transactions\}\end{aligned}$$

$$\begin{aligned}\mathit{TerminalTransactions} \triangleq\ & \{id \in all\_transactions : \\ & \quad \forall\, variant \in tx\_map[id] : \\ & \quad \forall\, d \in variant.ds : d \in participants\}\end{aligned}$$

$$\text{ASSUME } \forall\, id \in all\_transactions : \begin{array}{l} \lor\ id \in \mathit{TerminalTransactions} \\ \lor\ id \in \mathit{ContractTransactions}\end{array}$$

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

$$\begin{aligned}dependency\_map \triangleq\ & [dep\_id \in \\ & \quad \text{UNION } \{v.ds : v \in \text{UNION } \{tx\_map[id] : id \in \mathit{ContractTransactions}\}\} \\ & \mapsto \text{CHOOSE } id \in \mathit{ContractTransactions} : \\ & \qquad dep\_id \in \text{UNION } \{v.ds : v \in tx\_map[id]\}]\end{aligned}$$

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

$$\mathit{Contract} \triangleq \text{“Contract”}$$

$$\mathit{DstSet}(id, ds) \triangleq \text{IF } id \in \mathit{ContractTransactions} \text{ THEN } \{\mathit{Contract}\} \text{ ELSE } ds$$

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

Transaction also conflicts with itself

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

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

$$
\begin{array}{l}
DependencyBlock(id) \triangleq \\
\quad \text{CHOOSE } bn \in \text{DOMAIN } blocks : \\
\qquad dependency\_map[id] \in \{tx.id : tx \in blocks[bn]\}
\end{array}
$$

$$
\begin{array}{l}
Timelock(id, ds) \triangleq \\
\quad \text{LET } v \triangleq \text{CHOOSE } v \in tx\_map[id] : v.ds = ds \\
\quad\ \text{IN } \quad \text{IF } \text{"lk"} \in \text{DOMAIN } v \\
\qquad\qquad \text{THEN } v.lk \\
\qquad\qquad \text{ELSE } NoTimelock
\end{array}
$$

Transaction variants with different timelock types are not modelled

$$
\begin{array}{l}
\text{ASSUME } \forall\, id \in all\_transactions : \\
\qquad\qquad \forall\, v1 \in tx\_map[id] : \\
\qquad\qquad \forall\, v2 \in tx\_map[id] : \\
\qquad\qquad\quad \text{LET } t1 \triangleq Timelock(id, v1.ds) \\
\qquad\qquad\qquad\ \ t2 \triangleq Timelock(id, v2.ds) \\
\qquad\qquad\quad\ \text{IN } \quad t1.type = t2.type \lor NoTimelock \in \{t1, t2\}
\end{array}
$$

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

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

$$
\begin{array}{l}
\mathit{Deadline}(id,\ ds) \triangleq \\
\quad \text{LET } hs \triangleq \\
\qquad\qquad \text{UNION } \{ \\
\qquad\qquad\qquad \{ \\
\qquad\qquad\qquad\quad \mathit{TimelockExpirationHeight}(c\_id,\ v.ds) : v \in tx\_map[c\_id] \\
\qquad\qquad\qquad \} : c\_id \in \mathit{ConflictingSet}(id) \\
\qquad\qquad \} \\
\qquad c\_hs \triangleq \{h \in hs : h > \mathit{TimelockExpirationHeight}(id,\ ds)\} \\
\quad\ \text{IN} \quad \text{IF } c\_hs = \{\} \\
\qquad\quad \text{THEN } \mathit{UnreachableHeight} \quad \text{longest timelock (or no timelock)} \Rightarrow \text{no deadline} \\
\qquad\quad \text{ELSE } \mathit{Min}(c\_hs)
\end{array}
$$

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

This says 'Dependencies', but there's only one dependency possible for a transaction with current model

$$
\begin{array}{l}
\mathit{DependenciesMet}(id,\ ids) \triangleq \\
\quad id \in \text{DOMAIN } dependency\_map \Rightarrow dependency\_map[id] \in ids
\end{array}
$$

$$
\begin{aligned}
&IsSpendable(id, ss, ds, sender, other\_ids) \triangleq \\
&\quad \land \{\} = ConflictingSet(id) \cap other\_ids \\
&\quad \land DependenciesMet(id, other\_ids) \\
&\quad \land ss \subseteq SigsAvailable(id, ds, sender) \\
&\quad \land Len(blocks) \geq TimelockExpirationHeight(id, ds)
\end{aligned}
$$

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

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

$shared\_knowledge$ may not change here, callers need to check if they care

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

$$
\begin{aligned}
&ShareTransactions(ids, signer) \triangleq \\
&\quad \text{LET } Dst(id) \triangleq \text{CHOOSE } x \in DstSet(id, \{signer\}) : \text{TRUE} \\
&\qquad signer\_sigs \triangleq signers\_map[signer] \\
&\qquad txs \triangleq \text{UNION } \{ \\
&\qquad\qquad \{ \\
&\qquad\qquad\quad Tx(id, (v.ss \cap signer\_sigs) \setminus all\_secrets, \\
&\qquad\qquad\qquad v.ds, signer, Dst(id), \text{“direct”}) : \\
&\qquad\qquad\quad v \in tx\_map[id] \\
&\qquad\qquad \} : id \in ids \\
&\qquad \} \\
&\quad \text{IN } \land ShareKnowledge(txs) \\
&\qquad\quad \land shared\_knowledge' \neq shared\_knowledge
\end{aligned}
$$

not a new $knowledge \Rightarrow$ fail

Particpants shall not put transactions into $mempool$ past deadline, otherwise there may be contention and a chance for counterparty to take all

$$
IsSafeToSend(id, ds, sender) \triangleq Len(blocks) < Deadline(id, ds)
$$

$$
\begin{array}{l}
\mathit{SendTransactionToMempool}(id, variant, sender, to) \triangleq \\
\quad \land \mathit{IsSpendable}(id, variant.ss, variant.ds, sender, \mathit{SentTransactions}) \\
\quad \land \mathit{IsSafeToSend}(id, variant.ds, sender) \\
\quad \land \text{LET } tx \triangleq Tx(id, variant.ss, variant.ds, sender, to, \text{“mempool”}) \\
\qquad \text{IN} \quad \land mempool' = mempool \cup \{tx\} \\
\qquad \qquad \quad \land \mathit{ShareKnowledge}(\{tx\})
\end{array}
$$

Give tx directly to miner, bypassing global *mempool*
No *IsSafeToSend*() 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

$$
\begin{array}{l}
\mathit{SendTransactionToMiner}(id, variant, sender, to) \triangleq \\
\quad \land \mathit{STEALTHY\_SEND\_POSSIBLE} \\
\quad \land \mathit{IsSpendable}(id, variant.ss, variant.ds, sender, \\
\qquad \qquad \qquad \quad \mathit{NextBlockConfirmedTransactions}) \\
\quad \land next\_block' = \\
\qquad next\_block \cup \{Tx(id, variant.ss, variant.ds, sender, to, \text{“miner”})\}
\end{array}
$$

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

$$
\begin{array}{l}
\mathit{SendSomeTransaction}(ids, sender) \triangleq \\
\quad \exists\, id \in ids : \\
\quad \exists\, variant \in tx\_map[id] : \\
\quad \exists\, to \in \mathit{DstSet}(id, variant.ds \cap \{sender\}) : \\
\qquad \mathit{SendTransaction}(id, variant, sender, to)
\end{array}
$$

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

$$
\begin{aligned}
&\mathit{ContractIsLate} \triangleq \\
&\quad \land \mathit{Len}(\mathit{blocks}) \geq \mathit{BLOCKS\_PER\_DAY} \\
&\quad \land \forall\, \mathit{id} \in \mathit{all\_transactions} \setminus \mathit{excluded\_transactions} : \\
&\qquad \forall\, v \in \mathit{tx\_map}[\mathit{id}] : \\
&\qquad\quad \mathit{Len}(\mathit{blocks}) - \mathit{BLOCKS\_PER\_DAY} \geq \mathit{TimelockExpirationHeight}(\mathit{id}, v.\mathit{ds})
\end{aligned}
$$

Sharing secrets or keys has to occur before deadline to send *tx_success*

$$
\mathit{TooLateToShare} \triangleq \mathit{Len}(\mathit{blocks}) \geq \mathit{Deadline}(\mathit{tx\_success}, \mathit{SoleVariant}(\mathit{tx\_success}).\mathit{ds})
$$

Termination conditions

$$
\begin{aligned}
&\mathit{SwapSuccessful} \triangleq \\
&\quad \land \mathit{HasCustody}(\{\mathit{tx\_spend\_B}\}, \mathit{Alice}) \\
&\quad \land \mathit{HasCustody}(\{\mathit{tx\_spend\_A}, \mathit{tx\_spend\_success}, \\
&\qquad\qquad \mathit{tx\_spend\_timeout}, \mathit{tx\_spend\_revoke}\}, \mathit{Bob})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{SwapAborted} \triangleq \\
&\quad \land \lor \mathit{HasCustody}(\{\mathit{tx\_spend\_A}, \\
&\qquad\qquad \mathit{tx\_spend\_refund\_1}, \mathit{tx\_spend\_refund\_2}\}, \mathit{Alice}) \\
&\qquad \lor \exists\, \mathit{tx} \in \textsc{union}\ \mathit{Range}(\mathit{blocks}) : \land\, \mathit{tx}.\mathit{id} = \mathit{tx\_spend\_refund\_1} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \land\, \mathit{tx}.\mathit{ds} = \{\mathit{Alice}\} \\
&\quad \land \mathit{HasCustody}(\{\mathit{tx\_spend\_B}\}, \mathit{Bob})
\end{aligned}
$$

$$
\begin{aligned}
\mathit{SwapTimedOut} \triangleq\ & \land \mathit{tx\_spend\_timeout} \in \mathit{ConfirmedTransactions} \\
& \land \mathit{secretBob} \notin \mathit{signers\_map}[\mathit{Alice}]
\end{aligned}
$$

$$
\mathit{ContractFinished} \triangleq (\mathit{SwapSuccessful} \lor \mathit{SwapAborted} \lor \mathit{SwapTimedOut})
$$

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

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

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

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

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

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

Helper operators to declutter the action expressions

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

$$
\begin{array}{l}
\mathit{AliceAction} \triangleq \\
\quad \textsc{let}\ \mathit{Send}(\mathit{ids}) \triangleq \mathit{SendSomeTransaction}(\mathit{ids}, \mathit{Alice}) \\
\qquad\quad \mathit{Share}(\mathit{ids}) \triangleq \mathit{ShareTransactions}(\mathit{ids}, \mathit{Alice}) \\
\qquad\quad \mathit{OnlySafeToSend}(\mathit{ids}) \triangleq \\
\qquad\qquad \{ \\
\qquad\qquad\quad \mathit{id} \in \mathit{ids} : \\
\qquad\qquad\qquad \textsc{case}\ \mathit{id} = \mathit{tx\_refund\_1}\ \text{Do not send } \mathit{refund\_1} \text{ if } \mathit{tx\_success} \text{ was shared} \\
\qquad\qquad\qquad\qquad \rightarrow \mathit{tx\_success} \notin \{\mathit{tx.id} : \mathit{tx} \in \mathit{shared\_knowledge}\} \\
\qquad\qquad\qquad\quad \Box\ \mathit{secretAlice} \in \textsc{union}\ \{\mathit{v.ss} : v \in \mathit{tx\_map}[\mathit{id}]\} \\
\qquad\qquad\qquad\qquad \text{Once } \mathit{Alice} \text{ received } \mathit{secretBob}\text{, should never send out } \mathit{secretAlice} \\
\qquad\qquad\qquad\qquad \rightarrow \mathit{secretBob} \notin \mathit{signers\_map}[\mathit{Alice}] \\
\qquad\qquad\qquad\quad \Box\ \textsc{other} \rightarrow \textsc{true} \\
\qquad\qquad \} \\
\quad \textsc{in}\quad \lor \land \mathit{InPhase\_0} \\
\qquad\qquad\quad \land \mathit{Share}(\mathit{phase0\_to\_share\_Alice}) \\
\qquad\qquad\quad \land \mathit{NoSending} \land \mathit{NoKeysShared} \\
\qquad\qquad \lor \land \mathit{InPhase\_1} \\
\qquad\qquad\quad \land \mathit{Send}(\{\mathit{tx\_start\_A}\}) \\
\qquad\qquad\quad \land \mathit{NoKeysShared} \\
\qquad\qquad \lor \land \mathit{InPhase\_2} \\
\qquad\qquad\quad \land \textsc{false}\ \text{No specific actions} \\
\qquad\qquad \lor \land \mathit{InPhase\_3} \\
\qquad\qquad\quad \land \neg \mathit{HasCustody}(\mathit{TerminalTransactions}, \mathit{Alice})\ \ \mathit{Alice} \text{ gets } B \text{ or takes back A} \\
\qquad\qquad\quad \land \lor \land \mathit{secretBob} \in \mathit{signers\_map}[\mathit{Alice}]\ \ \text{Bob gave } \mathit{Alice} \text{ his secret} \\
\qquad\qquad\qquad\quad \land \mathit{sigAlice} \notin \mathit{signers\_map}[\mathit{Bob}]\ \ \mathit{Alice} \text{ did not yet gave } \mathit{Bob} \text{ her key} \\
\qquad\qquad\qquad\quad \land \neg \mathit{TooLateToShare} \\
\qquad\qquad\qquad\quad \land \mathit{signers\_map}' = \\
\qquad\qquad\qquad\qquad\quad [\mathit{signers\_map}\ \textsc{except}\ ![\mathit{Bob}] = \mathit{signers\_map}[\mathit{Bob}] \cup \{\mathit{sigAlice}\}] \\
\qquad\qquad\qquad\quad \land \mathit{NoSending} \land \mathit{NoKnowledgeShared} \\
\qquad\qquad\qquad \lor \land \mathit{tx\_refund\_1} \notin \mathit{SentTransactions} \\
\qquad\qquad\qquad\quad \land \neg \mathit{TooLateToShare} \\
\qquad\qquad\qquad\quad \land \mathit{Share}(\{\mathit{tx\_success}\})\ \ \mathit{refund\_1} \text{ not sent yet, can share} \\
\qquad\qquad\qquad\quad \land \mathit{NoSending} \land \mathit{NoKeysShared} \\
\qquad\qquad\qquad \lor \land \mathit{Send}(\mathit{OnlySafeToSend}(\mathit{all\_transactions})) \\
\qquad\qquad\qquad\quad \land \mathit{NoKeysShared}
\end{array}
$$

$$
\begin{array}{l}
\mathit{BobAction} \triangleq \\
\quad \text{LET } \mathit{Send}(\mathit{ids}) \triangleq \mathit{SendSomeTransaction}(\mathit{ids}, \mathit{Bob}) \\
\qquad\quad \mathit{Share}(\mathit{ids}) \triangleq \mathit{ShareTransactions}(\mathit{ids}, \mathit{Bob}) \\
\qquad\quad \mathit{tx\_success\_sigs} \triangleq \\
\qquad\qquad \mathit{SigsAvailable}(\mathit{tx\_success}, \mathit{SoleVariant}(\mathit{tx\_success}).\mathit{ds}, \mathit{Bob}) \\
\quad \text{IN } \quad \lor \land \mathit{InPhase\_0} \\
\qquad\qquad\quad \land \mathit{Share}(\mathit{phase0\_to\_share\_Bob}) \\
\qquad\qquad\quad \land \mathit{NoSending} \land \mathit{NoKeysShared} \\
\qquad\qquad \lor \land \mathit{InPhase\_1} \\
\qquad\qquad\quad \land \text{FALSE} \quad \text{No specific actions} \\
\qquad\qquad \lor \land \mathit{InPhase\_2} \\
\qquad\qquad\quad \land \mathit{Send}(\{\mathit{tx\_start\_B}\}) \\
\qquad\qquad\quad \land \mathit{NoKeysShared} \\
\qquad\qquad \lor \land \mathit{InPhase\_3} \quad \text{sign all transactions we can} \\
\qquad\qquad\quad \land \neg \mathit{HasCustody}(\mathit{TerminalTransactions}, \mathit{Bob}) \quad \text{Bob gets A or takes back } B \\
\qquad\qquad\quad \land \lor \land \mathit{sigAlice} \in \mathit{tx\_success\_sigs} \\
\qquad\qquad\qquad\qquad \text{If } \mathit{Bob} \text{ already knows } \mathit{secretAlice}\text{, he doesn't need to share } \mathit{secretBob} \\
\qquad\qquad\qquad\quad \land \mathit{secretAlice} \notin \mathit{tx\_success\_sigs} \\
\qquad\qquad\qquad\quad \land \mathit{secretBob} \notin \mathit{signers\_map}[\mathit{Alice}] \\
\qquad\qquad\qquad\quad \land \neg \mathit{TooLateToShare} \\
\qquad\qquad\qquad\quad \land \mathit{signers\_map}' = [\mathit{signers\_map} \quad \text{give } \mathit{secretBob} \text{ to } \mathit{Alice} \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{EXCEPT } ![\mathit{Alice}] = \mathit{signers\_map}[\mathit{Alice}] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \cup \{\mathit{secretBob}\}] \\
\qquad\qquad\qquad\quad \land \mathit{NoSending} \land \mathit{NoKnowledgeShared} \\
\qquad\qquad\qquad \lor \land \mathit{Send}(\mathit{all\_transactions}) \\
\qquad\qquad\qquad\quad \land \mathit{NoKeysShared}
\end{array}
$$

$$IsDeadlineOnNextBlock(id, ds) \triangleq Len(blocks) + 1 = Deadline(id, ds)$$

$$MempoolMonitorActionRequired \triangleq$$
$$\exists\, tx \in mempool : \land IsDeadlineOnNextBlock(tx.id, tx.ds)$$
$$\qquad\qquad\qquad\quad \land\; tx.id \notin NextBlockTransactions$$

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

$$MempoolMonitorAction \triangleq$$
$$\textsc{let}\; tx \triangleq \textsc{choose}\; tx \in mempool : IsDeadlineOnNextBlock(tx.id, tx.ds)$$
$$\qquad txs\_to\_bump \triangleq \{tx\} \cup \{dptx \in mempool : \land\; dptx.id = dependency\_map[tx.id] \land\; dptx.id \notin NextBlockTransactions\}$$
$$\textsc{in}\quad next\_block' = \{nbtx \in next\_block : \{\} = DependencyChain(nbtx.id) \cap \textsc{union}\; \{ConflictingSet(bmptx.id) : bmptx \in txs\_to\_bump\}\} \cup \{[bmptx\; \textsc{except}\; !.via = \text{“fee-bump”}] : bmptx \in txs\_to\_bump\}$$

(conflicting *txs* are expunged from *next_block*)

Miner action

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

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

$$
MinerAction \triangleq IncludeTxIntoBlock \lor MineTheBlock
$$

Invariants

$$
\begin{array}{l}
\mathit{TypeOK} \;\triangleq \\
\quad \textsc{let}\; \mathit{TxConsistent}(tx,\, vias) \;\triangleq \\
\qquad\qquad \land\; tx.id \in \mathit{all\_transactions} \\
\qquad\qquad \land\; tx.ss \subseteq \textsc{union}\; \{v.ss : v \in \mathit{tx\_map}[tx.id]\} \\
\qquad\qquad \land\; tx.ds \in \{v.ds : v \in \mathit{tx\_map}[tx.id]\} \\
\qquad\qquad \land\; tx.to \in \textsc{union}\; \{\mathit{DstSet}(tx.id,\, v.ds) : v \in \mathit{tx\_map}[tx.id]\} \\
\qquad\qquad \land\; tx.by \in \mathit{participants} \\
\qquad\qquad \land\; tx.via \in vias \\
\qquad\quad \mathit{AllSigsPresent}(tx) \;\triangleq\; \land\; tx.ss \in \{v.ss : v \in \mathit{tx\_map}[tx.id]\} \\
\qquad\quad \mathit{SigConsistent}(sig) \;\triangleq \\
\qquad\qquad \land\; sig.id \in \mathit{all\_transactions} \\
\qquad\qquad \land\; sig.s \in \mathit{all\_sigs} \\
\qquad\qquad \land\; sig.ds \subseteq \mathit{participants} \cup \textsc{domain}\; \mathit{dependency\_map} \\
\quad \textsc{in}\quad \land\; \forall\, tx \in \textsc{union}\; \mathit{Range}(\mathit{blocks}) : \\
\qquad\qquad \lor\; \land\; \mathit{TxConsistent}(tx,\, \{\text{“mempool”},\, \text{“miner”},\, \text{“fee-bump”}\}) \\
\qquad\qquad\quad\; \land\; \mathit{AllSigsPresent}(tx) \\
\qquad\qquad \lor\; \mathit{Print}(\langle \text{“\sim TypeOK blocks”},\, tx\rangle,\, \textsc{false}) \\
\qquad\quad \land\; \forall\, tx \in \mathit{next\_block} : \\
\qquad\qquad \lor\; \land\; \mathit{TxConsistent}(tx,\, \{\text{“mempool”},\, \text{“miner”},\, \text{“fee-bump”}\}) \\
\qquad\qquad\quad\; \land\; \mathit{AllSigsPresent}(tx) \\
\qquad\qquad \lor\; \mathit{Print}(\langle \text{“\sim TypeOK next\_block”},\, tx\rangle,\, \textsc{false}) \\
\qquad\quad \land\; \forall\, tx \in \mathit{mempool} : \\
\qquad\qquad \lor\; \land\; \mathit{TxConsistent}(tx,\, \{\text{“mempool”}\}) \\
\qquad\qquad\quad\; \land\; \mathit{AllSigsPresent}(tx) \\
\qquad\qquad \lor\; \mathit{Print}(\langle \text{“\sim TypeOK mempool”},\, tx\rangle,\, \textsc{false}) \\
\qquad\quad \land\; \forall\, tx \in \mathit{shared\_knowledge} : \\
\qquad\qquad \lor\; \mathit{TxConsistent}(tx,\, \{\text{“mempool”},\, \text{“miner”},\, \text{“fee-bump”},\, \text{“direct”}\}) \\
\qquad\qquad \lor\; \mathit{Print}(\langle \text{“\sim TypeOK shared\_knowledge”},\, tx\rangle,\, \textsc{false}) \\
\qquad\quad \land\; \forall\, p \in \textsc{domain}\; \mathit{signers\_map} : \\
\qquad\qquad \lor\; \land\; p \in \mathit{participants} \\
\qquad\qquad\quad\; \land\; \forall\, sig \in \mathit{signers\_map}[p] : sig \in \mathit{all\_sigs} \\
\qquad\qquad \lor\; \mathit{Print}(\langle \text{“\sim TypeOK signers\_map”},\, p\rangle,\, \textsc{false}) \\
\qquad\quad \land\; \mathit{excluded\_transactions} \subseteq \mathit{all\_transactions}
\end{array}
$$

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

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

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

$$
\begin{array}{l}
\mathit{NoSingleParticipantTakesAll} \triangleq \\
\quad \forall\, p \in \mathit{participants} : \\
\qquad \textsc{let}\ \mathit{txs\_to\_p} \triangleq \{\mathit{tx} \in \textsc{union}\ \mathit{Range}(\mathit{blocks}) : \mathit{tx.to} = p\} \\
\qquad \textsc{in}\quad \mathit{Cardinality}(\{\mathit{tx.id} : \mathit{tx} \in \mathit{txs\_to\_p}\}) \leq 1
\end{array}
$$

$$
\begin{array}{l}
\mathit{NoUnsafeTransactionPublishing} \triangleq \\
\quad \forall\, \mathit{tx} \in \mathit{mempool} : \mathit{IsSafeToSend}(\mathit{tx.id}, \mathit{tx.ds}, \mathit{tx.by})
\end{array}
$$

$$
\begin{array}{l}
\mathit{TransactionTimelocksEnforced} \triangleq \\
\quad \neg\exists\, \mathit{tx} \in \mathit{next\_block} : \mathit{Len}(\mathit{blocks}) < \mathit{TimelockExpirationHeight}(\mathit{tx.id}, \mathit{tx.ds})
\end{array}
$$

$$
\begin{array}{l}
\mathit{CleanStateOnContractFinish} \triangleq \\
\quad \mathit{ContractFinished} \Rightarrow \\
\qquad \land\ \mathit{mempool} = \{\} \\
\qquad \land\ \mathit{next\_block} = \{\} \\
\qquad \land\ \neg\textsc{enabled}\ \mathit{AliceAction} \lor \mathit{Print}(\text{“AliceAction is enabled”}, \textsc{false}) \\
\qquad \land\ \neg\textsc{enabled}\ \mathit{BobAction}\ \ \lor \mathit{Print}(\text{“BobAction is enabled”}, \textsc{false})
\end{array}
$$

$$
\mathit{ContractFinishesBeforeTooLate} \triangleq \mathit{ContractIsLate} \Rightarrow \mathit{ContractFinished}
$$

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

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

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

Can use this to manually check that any transaction
can eventually be confirmed (much faster than via temporal properties)

$$
\begin{aligned}
&\land tx\_start\_A \notin ConfirmedTransactions\\
&\land tx\_start\_B \notin ConfirmedTransactions\\
&\land tx\_success \notin ConfirmedTransactions\\
&\land tx\_refund\_1 \notin ConfirmedTransactions\\
&\land tx\_revoke \notin ConfirmedTransactions\\
&\land tx\_refund\_2 \notin ConfirmedTransactions\\
&\land tx\_timeout \notin ConfirmedTransactions\\
&\land tx\_spend\_A \notin ConfirmedTransactions\\
&\land tx\_spend\_B \notin ConfirmedTransactions\\
&\land tx\_spend\_success \notin ConfirmedTransactions\\
&\land tx\_spend\_refund\_1 \notin ConfirmedTransactions\\
&\land tx\_spend\_revoke \notin ConfirmedTransactions\\
&\land tx\_spend\_refund\_2 \notin ConfirmedTransactions\\
&\land tx\_spend\_timeout \notin ConfirmedTransactions
\end{aligned}
$$

Temporal properties

$$
\begin{aligned}
&RevokeLeadsToAbortOrTimeout \triangleq\\
&\quad tx\_revoke \in NextBlockTransactions \leadsto \Box\Diamond(SwapAborted \lor SwapTimedOut)
\end{aligned}
$$

$$ContractAlwaysEventuallyFinished \triangleq \Box\Diamond ContractFinished$$

much faster to check manually via counterexample for each transaction

$$
\begin{aligned}
&EachTransactionEventuallyConfirmed \triangleq\\
&\quad \forall\, id \in all\_transactions : \Diamond(id \in ConfirmedTransactions)
\end{aligned}
$$

High-level spec

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

$$
\begin{aligned}
&\mathit{Next} \triangleq \\
&\quad \lor\ \mathit{AliceAction} \qquad\qquad\qquad\quad \land\ \textsc{unchanged}\ \mathit{blockState} \\
&\quad \lor\ \mathit{BobAction} \qquad\qquad\qquad\quad\ \ \land\ \textsc{unchanged}\ \mathit{blockState} \\
&\quad \lor\ \textsc{if}\ \mathit{MempoolMonitorActionRequired} \\
&\qquad\ \ \textsc{then}\ \mathit{MempoolMonitorAction} \land\ \textsc{unchanged}\ \mathit{allExceptNextBlock} \\
&\qquad\ \ \textsc{else}\ \ \mathit{MinerAction} \qquad\qquad\quad\ \land\ \textsc{unchanged}\ \mathit{signers\_map} \\
&\quad \lor\ \mathit{ContractFinished} \qquad\qquad\qquad\ \land\ \textsc{unchanged}\ \mathit{fullState}
\end{aligned}
$$

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