1 Cryptographic primitives

We rely on a public key signing scheme for verification of spending.

Types & functions

$$\mathsf{SKey}\ \mathsf{VKey}\ \mathsf{Sig}\ \mathsf{Ser} : \mathsf{Set}$$
$$\mathsf{isKeyPair} : \mathsf{SKey} \to \mathsf{VKey} \to \mathsf{Set}$$
$$\mathsf{isSigned} : \mathsf{VKey} \to \mathsf{Ser} \to \mathsf{Sig} \to \mathsf{Set}$$
$$\mathsf{sign} : \mathsf{SKey} \to \mathsf{Ser} \to \mathsf{Sig}$$

$$\mathsf{KeyPair} = \Sigma[\ sk \in \mathsf{SKey}\ ]\ \Sigma[\ vk \in \mathsf{VKey}\ ]\ \mathsf{isKeyPair}\ sk\ vk$$

Property of signatures

$$((sk\ ,\ vk\ ,\ \_) : \mathsf{KeyPair})\ (d : \mathsf{Ser})\ (\sigma : \mathsf{Sig}) \to \mathsf{sign}\ sk\ d \equiv \sigma \to \mathsf{isSigned}\ vk\ d\ \sigma$$

Figure 1: Definitions for the public key signature scheme

2 Addresses

We define credentials and various types of addresses here.

Abstract types

$$\begin{array}{l} \mathit{Network} \\ \mathit{KeyHash} \\ \mathit{ScriptHash} \end{array}$$

Derived types

$$\mathsf{Credential} = \mathit{KeyHash} \uplus \mathit{ScriptHash}$$

```
record BaseAddr : Set where
  field net   : Network
        pay   : Credential
        stake : Credential

record BootstrapAddr : Set where
  field net       : Network
        pay       : Credential
        attrsSize : ℕ

record RwdAddr : Set where
  field net   : Network
        stake : Credential

Addr = BaseAddr ⊎ BootstrapAddr

VKeyBaseAddr        = Σ[ addr ∈ BaseAddr      ] isVKey (BaseAddr.pay      addr)
VKeyBootstrapAddr   = Σ[ addr ∈ BootstrapAddr ] isVKey (BootstrapAddr.pay addr)

ScriptBaseAddr      = Σ[ addr ∈ BaseAddr      ] isScript (BaseAddr.pay      addr)
ScriptBootstrapAddr = Σ[ addr ∈ BootstrapAddr ] isScript (BootstrapAddr.pay addr)

VKeyAddr = VKeyBaseAddr ⊎ VKeyBootstrapAddr
ScriptAddr = ScriptBaseAddr ⊎ ScriptBootstrapAddr
```

Helper functions

```
payCred    : Addr → Credential
netId      : Addr → Network
isVKeyAddr : Addr → Set
```

Figure 2: Definitions used in Addresses

3 Scripts

We define Timelock scripts here. They can verify the presence of keys and whether a transaction happens in a certain slot interval. These scripts are executed as part of the regular witnessing.

$$
\begin{array}{llll}
\multicolumn{4}{l}{\mathsf{data}\ \mathsf{Timelock} : \mathsf{Set}\ \mathsf{where}} \\
\quad \mathsf{RequireAllOf} & : \mathsf{List}\ \mathsf{Timelock} & \to & \mathsf{Timelock} \\
\quad \mathsf{RequireAnyOf} & : \mathsf{List}\ \mathsf{Timelock} & \to & \mathsf{Timelock} \\
\quad \mathsf{RequireMOf} & : \mathbb{N} \to \mathsf{List}\ \mathsf{Timelock} & \to & \mathsf{Timelock} \\
\quad \mathsf{RequireSig} & : \mathit{KeyHash} & \to & \mathsf{Timelock} \\
\quad \mathsf{RequireTimeStart} & : \mathit{Slot} & \to & \mathsf{Timelock} \\
\quad \mathsf{RequireTimeExpire} & : \mathit{Slot} & \to & \mathsf{Timelock}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{data}\ \mathsf{evalTimelock}\ (\mathit{khs} : \mathbb{P}\ \mathit{KeyHash})\ (I : \mathsf{Maybe}\ \mathit{Slot} \times \mathsf{Maybe}\ \mathit{Slot}) : \mathsf{Timelock} \to \mathsf{Set}\ \mathsf{where} \\
\quad \mathsf{evalAll} : \mathsf{All}\ (\mathsf{evalTimelock}\ \mathit{khs}\ I)\ \mathit{ss} \to \mathsf{evalTimelock}\ \mathit{khs}\ I\ (\mathsf{RequireAllOf}\ \mathit{ss}) \\
\quad \mathsf{evalAny} : \mathsf{Any}\ (\mathsf{evalTimelock}\ \mathit{khs}\ I)\ \mathit{ss} \to \mathsf{evalTimelock}\ \mathit{khs}\ I\ (\mathsf{RequireAnyOf}\ \mathit{ss}) \\
\quad \mathsf{evalMOf} : \mathit{ss'}\ \mathsf{S.}\!\subseteq\ \mathit{ss} \to \mathsf{All}\ (\mathsf{evalTimelock}\ \mathit{khs}\ I)\ \mathit{ss'} \to \mathsf{evalTimelock}\ \mathit{khs}\ I\ (\mathsf{RequireMOf}\ (\mathsf{length}\ \mathit{ss'})\ \mathit{ss}) \\
\quad \mathsf{evalSig} : x \in \mathit{khs} \to \mathsf{evalTimelock}\ \mathit{khs}\ I\ (\mathsf{RequireSig}\ x) \\
\quad \mathsf{evalTSt} : \mathsf{proj}_1\ I \equiv \mathsf{just}\ l \to a \leq l \to \mathsf{evalTimelock}\ \mathit{khs}\ I\ (\mathsf{RequireTimeStart}\ a) \\
\quad \mathsf{evalTEx} : \mathsf{proj}_2\ I \equiv \mathsf{just}\ r \to r \leq a \to \mathsf{evalTimelock}\ \mathit{khs}\ I\ (\mathsf{RequireTimeStart}\ a)
\end{array}
$$

Figure 3: Timelock scripts and their evaluation

4 Protocol parameters

```
ProtVer = ℕ × ℕ

record PParams : Set where
  field a             : ℕ
        b             : ℕ
        maxBlockSize  : ℕ
        maxTxSize     : ℕ
        maxHeaderSize : ℕ
        maxValSize    : ℕ
        minUtxOValue  : Coin
        poolDeposit   : Coin
        Emax          : Epoch
        pv            : ProtVer
```

Figure 4: Definitions used for protocol parameters

5 Transactions

Transactions are defined in Figure 5. A transaction is made up of three pieces:

- A set of transaction inputs. This derived type identifies an output from a previous transaction. It consists of a transaction id and an index to uniquely identify the output.
- An indexed collection of transaction outputs. The TxOut type is an address paired with a coin value.
- A transaction fee. This value will be added to the fee pot.

Finally, txid computes the transaction id of a given transaction. This function must produce a unique id for each unique transaction body. **We assume that txid is injective.**

$$\mathrm{getValue} : \mathrm{TxOut} \to \mathrm{Value}$$
$$\mathrm{getValue}\ (\mathit{fst}\ ,\ \mathit{snd}) = \mathit{snd}$$

$$\mathrm{txinsVKey} : \mathbb{P}\ \mathrm{TxIn} \to \mathrm{UTxO} \to \mathbb{P}\ \mathrm{TxIn}$$
$$\mathrm{txinsVKey}\ \mathit{txins}\ \mathit{utxo} = \mathit{txins} \cap \mathrm{dom}\ ((\mathit{utxo}\ |\text{\^{}}'\ \text{to-sp}\ (\mathrm{isVKeyAddr?} \circ \mathrm{proj}_1))\ ^{s})$$

Abstract types

$$\text{Ix TxId AuxiliaryData} : \text{Set}$$

Derived types

$$
\begin{aligned}
\text{TxIn} &= \text{TxId} \times \text{Ix}\\
\text{TxOut} &= \text{Addr} \times \text{Value}\\
\text{UTxO} &= \text{TxIn} \rightharpoonup \text{TxOut}\\
\text{Wdrl} &= \text{RwdAddr} \rightharpoonup \text{Coin}
\end{aligned}
$$

$$
\begin{aligned}
&\text{ProposedPPUpdates} = \text{KeyHash} \rightharpoonup \text{PParamsUpdate}\\
&\text{Update} = \text{ProposedPPUpdates} \times \text{Epoch}
\end{aligned}
$$

Transaction types

```
record TxBody : Set where
  field txins     : ℙ TxIn
        txouts    : Ix ⇀ TxOut
        --txcerts  : List DCert
        mint      : Value
        txfee     : Coin
        txvldt    : Maybe Slot × Maybe Slot
        txwdrls   : Wdrl
        txup      : Maybe Update
        txADhash  : Maybe ADHash
        netwrk    : Maybe Network
        txsize    : ℕ
        txid      : TxId

record TxWitnesses : Set where
  field vkSigs  : VKey ⇀ Sig
        scripts : ℙ Script

record Tx : Set where
  field body : TxBody
        wits : TxWitnesses
        txAD : Maybe AuxiliaryData
```

Abstract functions

Figure 5: Definitions used in the UTxO transition system

6 UTxO

6.1 Accounting

Figure 6 defines functions needed for the UTxO transition system. Figure 7 defines the types needed for the UTxO transition system. The UTxO transition system is given in Figure 8.

- The function outs creates the unspent outputs generated by a transaction. It maps the transaction id and output index to the output.
- The balance function calculates sum total of all the coin in a given UTxO.

```
outs : TxBody → UTxO
outs tx = mapKeys (txid tx ,_) (λ where refl → refl) $ txouts tx

balance : UTxO → Value
balance utxo = Σᵐ[ x ← utxo ᶠᵐ ] proj₂ (proj₂ x)

cbalance : UTxO → Coin
cbalance utxo = coin (balance utxo)

minfee : PParams → TxBody → Coin
minfee pp tx = a * txsize tx + b
  where open PParams pp

-- need to add withdrawals to consumed
consumed : PParams → UTxO → TxBody → Value
consumed pp utxo txb = balance (utxo | txins txb)
                       +ᵛ mint txb
                     --+ inject (wbalance (txwdrls txb) + keyRefunds pp txb)

-- need to add deposits to produced
produced : PParams → UTxO → TxBody → Value
produced pp utxo txb = balance (outs txb)
                       +ᵛ inject (txfee txb)
                     --+ totalDeposits pp stpools (txcerts txb))

-- this has to be a type definition for inference to work
data inInterval (slot : Slot) : (Maybe Slot × Maybe Slot) → Set where
  both  : ∀ {l r} → l ≤ˢ slot × slot ≤ˢ r → inInterval slot (just l , just r)
  lower : ∀ {l}   → l ≤ˢ slot             → inInterval slot (just l , nothing)
  upper : ∀ {r}   → slot ≤ˢ r             → inInterval slot (nothing , just r)
  none  :                                   inInterval slot (nothing , nothing)
```

Figure 6: Functions used in UTxO rules

UTxO environment

```
record UTxOEnv : Set where
  field slot : Slot
        pparams : PParams
```

UTxO states

```
record UTxOState : Set where
  constructor ⟦_,_⟧ᵘ
  field utxo : UTxO
        fees : Coin
```

UTxO transitions

```
_⊢_⇀⦇_,UTXO⦈_ : UTxOEnv → UTxOState → TxBody → UTxOState → Set
```

Figure 7: UTxO transition-system types

```
UTXO-inductive :
  ∀ {Γ} {s} {tx}
  → let slot = UTxOEnv.slot Γ
        pp = UTxOEnv.pparams Γ
        utxo = UTxOState.utxo s
        fees = UTxOState.fees s
    in
  txins tx ≢ ∅
  → inInterval slot (txvldt tx)
  → txins tx ⊆ dom (utxo ˢ)
  → let f = txfee tx in minfee pp tx ≤ f
  → consumed pp utxo tx ≡ produced pp utxo tx
  → coin (mint tx) ≡ 0

{- these break deriveComputational but don't matter at the moment
    →   txout → txout   proj₁ (txouts tx)
              → (getValue (proj₂ txout))   (inject (utxoEntrySize (proj₂ txout) * PPara

    →   txout → txout   proj₁ (txouts tx)
              → (serSize (getValue (proj₂ txout))) ≤ PParams.maxValSize pp
-}

  -- PPUP
  -- these fail with some reduceDec error
  -- → All (  { (inj₂ a , _) → BootstrapAddr.attrsSize a ≤ 64 ; _ →   }) (range ((txouts
  -- → All (  a → netId (proj₁ a)   networkId) (range ((txouts tx)  ))
  -- → All (  a → RwdAddr.net a   networkId) (dom ((txwdrls tx)  ))
  → txsize tx ≤ PParams.maxTxSize pp
  -- Add Deposits
  ────────────────────────────────────────────
  Γ
    ⊢ s
    ⇀⦇ tx ,UTXO⦈
    ⟦ (utxo ∣ txins tx ᶜ) ∪ˡ outs tx , fees + f ⟧ᵘ
```

Figure 8: UTXO inference rules

6.2 Witnessing

```
witsVKeyNeeded : UTxO → TxBody → ℙ KeyHash
witsVKeyNeeded utxo txb =
  mapPartial ((λ { (inj₁ kh) → just kh ; _ → nothing }) ∘ payCred ∘ proj₁) ((utxo ˢ) ⟪$⟫ txins txb)

scriptsNeeded : UTxO → TxBody → ℙ ScriptHash
scriptsNeeded utxo txb =
  mapPartial ((λ { (inj₂ sh) → just sh ; _ → nothing }) ∘ payCred ∘ proj₁) ((utxo ˢ) ⟪$⟫ txins txb)

scriptsP1 : TxWitnesses → ℙ P1Script
scriptsP1 txw = mapPartial (λ { (inj₁ s) → just s ; _ → nothing }) (scripts txw)
```

Figure 9: Functions used for witnessing

```
_⊢_⇀⦇_,UTXOW⦈_ : UTxOEnv → UTxOState → Tx → UTxOState → Set
```

Figure 10: UTxOW transition-system types

```
UTXOW-inductive :
  ∀ {Γ} {s} {tx} {s'}
  → let utxo = UTxOState.utxo s
        txb = body tx
        txw = wits tx
        witsKeyHashes = map hash (dom (vkSigs txw ˢ))
        witsScriptHashes = map hash (scripts txw)
     in
  All (λ where (vk , σ) → isSigned vk (txidBytes (txid txb)) σ) (vkSigs txw ˢ)
  → All (validP1Script witsKeyHashes (txvldt txb)) (scriptsP1 txw)
  → witsVKeyNeeded utxo txb ⊆ witsKeyHashes
  → scriptsNeeded utxo txb ≡ᵉ witsScriptHashes
  -- TODO: check genesis signatures
  → txADhash txb ≡ M.map hash (txAD tx)
  → Γ ⊢ s ⇀⦇ txb ,UTXO⦈ s'
  ────────────────────────────────────────
  Γ ⊢ s ⇀⦇ tx ,UTXOW⦈ s'
```

Figure 11: UTXOW inference rules

7 Update Proposal Mechanism

```
GenesisDelegation = KeyHash ⇸ (KeyHash × KeyHash)

record PPUpdateState : Set where
  field pup  : ProposedPPUpdates
        fpup : ProposedPPUpdates

record PPUpdateEnv : Set where
  field slot      : Slot
        pparams   : PParams
        genDelegs : GenesisDelegation
```

Figure 12: PPUP types

```
data pvCanFollow : ProtVer → ProtVer → Set where
  canFollowMajor : pvCanFollow (m , n) (m ℕ.+ 1 , 0)
  canFollowMinor : pvCanFollow (m , n) (m , n ℕ.+ 1)

viablePParams : PParams → Set
viablePParams pp = ⊤ -- TODO: block size check

isViableUpdate : PParams → PParamsUpdate → Set
isViableUpdate pp pup with applyUpdate pp pup
... | pp' = pvCanFollow (PParams.pv pp) (PParams.pv pp') × viablePParams pp'
```

Figure 13: Definitions for PPUP

$$\mathsf{PPUpdateEmpty} : \Gamma \vdash s \rightharpoonup\llparenthesis\ \mathsf{nothing}\ ,\mathsf{PPUP}\rrparenthesis\ s$$

$$\begin{array}{l}
\mathsf{PPUpdateCurrent} : \mathsf{let\ open\ PPUpdateEnv}\ \Gamma\ \mathsf{in} \\
\quad \mathsf{dom}\ (pup\ ^{\mathsf{s}}) \subseteq \mathsf{dom}\ (\mathsf{genDelegs}\ ^{\mathsf{s}}) \\
\quad \to \mathsf{All}\ (\mathsf{isViableUpdate\ pparams})\ (\mathsf{range}\ (pup\ ^{\mathsf{s}})) \\
\quad \to (\mathsf{slot} + (2 * \mathsf{StabilityWindow})) <^{s} \mathsf{firstSlot}\ (\mathsf{suc}^{e}\ (\mathsf{epoch\ slot})) \\
\quad \to \mathsf{epoch\ slot} \equiv e \\
\hline
\quad \Gamma \vdash \mathsf{record}\ \{\ \mathsf{pup} = pup\ \ ;\ \mathsf{fpup} = fpup\ \ \} \rightharpoonup\llparenthesis\ \mathsf{just}\ (pup\ ,\ e)\ ,\mathsf{PPUP}\rrparenthesis \\
\qquad\quad \mathsf{record}\ \{\ \mathsf{pup} = pup \cup^{ml} pup\ \ ;\ \mathsf{fpup} = fpup\ \ \}
\end{array}$$

$$\begin{array}{l}
\mathsf{PPUpdateFuture} : \mathsf{let\ open\ PPUpdateEnv}\ \Gamma\ \mathsf{in} \\
\quad \mathsf{dom}\ (pup\ ^{\mathsf{s}}) \subseteq \mathsf{dom}\ (\mathsf{genDelegs}\ ^{\mathsf{s}}) \\
\quad \to \mathsf{All}\ (\mathsf{isViableUpdate\ pparams})\ (\mathsf{range}\ (pup\ ^{\mathsf{s}})) \\
\quad \to (\mathsf{slot} + (2 * \mathsf{StabilityWindow})) \geq^{s} \mathsf{firstSlot}\ (\mathsf{suc}^{e}\ (\mathsf{epoch\ slot})) \\
\quad \to \mathsf{suc}^{e}\ (\mathsf{epoch\ slot}) \equiv e \\
\hline
\quad \Gamma \vdash \mathsf{record}\ \{\ \mathsf{pup} = pup\ \ ;\ \mathsf{fpup} = fpup\ \ \} \rightharpoonup\llparenthesis\ \mathsf{just}\ (pup\ ,\ e)\ ,\mathsf{PPUP}\rrparenthesis \\
\qquad\quad \mathsf{record}\ \{\ \mathsf{pup} = pup\ \ ;\ \mathsf{fpup} = pup \cup^{ml} fpup\ \ \}
\end{array}$$

Figure 14: PPUP inference rules

8 Ledger State Transition

```
-- Only include accounting & governance info for now
record LEnv : Set where
  constructor ⟦_,_,_⟧ˡᵉ
  field slot : Slot
        --txix : Ix
        pparams : PParams
        --acnt : Acnt
        genDelegs : GenesisDelegation -- part of DPState

record LState : Set where
  constructor ⟦_,_⟧ˡ
  field utxoSt : UTxOState
        ppup : PPUpdateState
        --dpstate : DPState
```

Figure 15: Types for the LEDGER transition system

```
LEDGER : let open LState s in
  record { LEnv Γ } ⊢ utxoSt ⇀⦇ tx ,UTXOW⦈ utxoSt'
  → record { LEnv Γ } ⊢ ppup ⇀⦇ txup (body tx) ,PPUP⦈ ppup'
  -- DELEGS
  ────────────────────────────────
  Γ ⊢ s ⇀⦇ tx ,LEDGER⦈ ⟦ utxoSt' , ppup' ⟧ˡ
```

Figure 16: LEDGER transition system

$$\text{LEDGERS-base} : \Gamma \vdash s \rightharpoonup\llparenthesis\ [] \ \text{,LEDGERS}\rrparenthesis\ s$$

$$\text{LEDGERS-ind} : \forall \{txs\}$$

$$\rightarrow \Gamma \vdash s \rightharpoonup\llparenthesis\ tx \ \text{,LEDGER}\rrparenthesis\ s'$$

$$\rightarrow \Gamma \vdash s' \rightharpoonup\llparenthesis\ txs \ \text{,LEDGERS}\rrparenthesis\ s''$$

$$\overline{\Gamma \vdash s \rightharpoonup\llparenthesis\ tx :: txs \ \text{,LEDGERS}\rrparenthesis\ s''}$$

Figure 17: LEDGERS transition system

9 Protocol Parameters Update

```
record NewPParamState : Set where
  constructor ⟦_,_⟧ⁿᵖ
  field pparams : PParams
        ppup    : PPUpdateState

updatePPUp : PParams → PPUpdateState → PPUpdateState
updatePPUp pparams record { fpup = fpup } with allᵇ (isViableUpdate? pparams) (range (fpup ˢ))
... | false = record { pup = ∅ᵐ   ; fpup = ∅ᵐ }
... | true  = record { pup = fpup ; fpup = ∅ᵐ }

votedValue : ProposedPPUpdates → PParams → ℕ → Maybe PParamsUpdate
votedValue pup pparams quorum =
  case any? (λ u → lengthˢ ((pup |^ fromList [ u ]) ˢ) ≥? quorum) (range (pup ˢ)) of λ where
    (no _)        → nothing
    (yes (u , _)) → just u
```

Figure 18: Types and functions for the NEWPP transition system

```
NEWPP-Accept : ∀ {Γ} → let open NewPParamState s; newpp = applyUpdate pparams upd in
  viablePParams newpp
  ────────────────────────────────
  Γ ⊢ s ⇀⦇ just upd ,NEWPP⦈ ⟦ newpp , updatePPUp newpp ppup ⟧ⁿᵖ

NEWPP-Reject : ∀ {Γ} → let open NewPParamState s in
  Γ ⊢ s ⇀⦇ nothing ,NEWPP⦈ ⟦ pparams , updatePPUp newpp ppup ⟧ⁿᵖ
```

Figure 19: NEWPP transition system

10 Blockchain layer

```
record Acnt : Set where
  field treasury : Coin
        reserves : Coin

record NewEpochState : Set where
  constructor ⟦_,_,_,_,_⟧ⁿᵉ
  field lastEpoch : Epoch
        acnt      : Acnt
        ls        : LState
        prevPP    : PParams
        pparams   : PParams

record ChainState : Set where
  field newEpochState : NewEpochState
        genDelegs     : GenesisDelegation

record Block : Set where
  field ts : List Tx
        slot : Slot
```

Figure 20: Definitions for the NEWEPOCH and CHAIN transition systems

```
NEWEPOCH-New : let
  open NewEpochState nes
  open LState ls
  pup = PPUpdateState.pup ppup
  acnt' = record acnt { treasury = Acnt.treasury acnt + UTxOState.fees utxoSt }
  ls' = record ls { ppup = ppup' ; utxoSt = record utxoSt { fees = 0 } }
  in
  e ≡ sucᵉ lastEpoch
  → _ ⊢ ⟦ pparams , ppup ⟧ⁿᵖ ⇀⦇ votedValue pup pparams Quorum ,NEWPP⦈ ⟦ pparams' , ppup' ⟧ⁿᵖ
  ────────────────────────────────────────────────
  _ ⊢ nes ⇀⦇ e ,NEWEPOCH⦈ ⟦ e , acnt' , ls' , pparams , pparams' ⟧ⁿᵉ

NEWEPOCH-Not-New : let open NewEpochState nes in
  e ≢ sucᵉ lastEpoch
  ────────────────────────────────────────────────
  _ ⊢ nes ⇀⦇ e ,NEWEPOCH⦈ nes
```

Figure 21: NEWEPOCH transition system

```
CHAIN : let open ChainState s; open Block b; open NewEpochState in
  _ ⊢ newEpochState ⇀⦇ epoch slot ,NEWEPOCH⦈ nes
  → ⟦ slot , pparams nes , genDelegs ⟧ˡᵉ ⊢ ls nes ⇀⦇ ts ,LEDGERS⦈ ls'
  ────────────────────────────────────────────────
  _ ⊢ s ⇀⦇ b ,CHAIN⦈ record s { newEpochState = record nes { ls = ls' } }
```

Figure 22: CHAIN transition system

11 Properties

11.1 UTxO

Property 11.1 (Preserve Balance) *For all* $env \in \text{UTxOEnv}$*,* $utxo, utxo' \in \text{UTxO}$*,* $fee, fee' \in \text{Coin}$ *and* $tx \in \text{TxBody}$*, if* txid *tx* ∉ map proj_1 (dom (*utxo* ˢ)) *and* *Γ* ⊢ ⟦ *utxo* , *fee* ⟧ᵘ ⇀⦇ *tx* ,UTXO⦈ ⟦ *utxo'* , *fee'* ⟧ᵘ *then*

> getCoin ⟦ *utxo* , *fee* ⟧ᵘ ≡ getCoin ⟦ *utxo'* , *fee'* ⟧ᵘ

Here, we state the fact that the UTxO relation is computable. This just follows from our automation.

```
UTXO-step : UTxOEnv → UTxOState → TxBody → Maybe UTxOState
UTXO-step = compute Computational-UTXO

UTXO-step-computes-UTXO :
  UTXO-step Γ utxoState tx ≡ just utxoState' ⇔ Γ ⊢ utxoState ⇀⦇ tx ,UTXO⦈ utxoState'
UTXO-step-computes-UTXO = ≡-just⇔STS Computational-UTXO
```

Figure 23: Computing the UTXO transition system