1 Transactions

Transactions are defined in Figure 1. 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.**

Abstract types

```
Ix   -- index
TxId -- transaction id
Addr -- address
```

Derived types

$$\mathsf{Coin} = \mathbb{N}$$
$$\mathsf{TxIn} = \mathit{TxId} \times \mathit{Ix}$$
$$\mathsf{TxOut} = \mathit{Addr} \times \mathsf{Coin}$$
$$\mathsf{UTxO} = \mathsf{TxIn} \mapsto \mathsf{TxOut}$$

Transaction types

```
record Tx : Set where
  field
    txins : ℙ TxIn
    txouts : Ix ↦ TxOut
    txfee : Coin
```

Abstract functions

```
txid : Tx ↪ TxId -- an injective function
```

Figure 1: Definitions used in the UTxO transition system

2 UTxO

Figure 2 defines functions needed for the UTxO transition system. Figure 3 defines the types needed for the UTxO transition system. The UTxO transition system is given in Figure 4.

- 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 : Tx → UTxO
outs tx = mapKeys (txid ⟨$⟩ tx ,_) $ txouts tx

balance : UTxO → Coin
balance utxo = Σ[ v ← utxo ] proj₂ (proj₂ v)
```

Figure 2: Functions used in UTxO rules

UTxO environment

```
UTxOEnv = Coin -- minimum fee
```

UTxO states

```
UTxOState = UTxO -- UTxO
          × Coin -- fee pot
```

UTxO transitions

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

Figure 3: UTxO transition-system types

Property 2.1 (Preserve Balance) *For all minFee* ∈ UTxOEnv*, utxo, utxo′* ∈ UTxOState*, and tx* ∈ Tx*, if utxo* ∩ outs *tx* ≡ ∅ *and minFee* ⊢ (*utxo* , *fee*) ⇀⦇ *tx* ,UTXO⦈ (*utxo'* , *fee'*) *then*

```
balance utxo + fee ≡ balance utxo' + fee'
```

```
UTXO-inductive :
    txins tx ⊆ dom utxo
  → let f = txfee tx in minFee ≤ f
  → balance (txins tx ◃ utxo) ≡ balance (outs tx) + f
  ────────────────────────────────────────────────
  minFee
  ⊢ ⟦ utxo , fees ⟧
  ⇀⦇ tx ,UTXO⦈
    ⟦ (txins tx ⋪ utxo) ∪ outs tx , fees + txfee tx ⟧
```

Figure 4: UTXO inference rules

Note that this is not a function, but a relation. To make this definition executable, we need to define a function that computes the transition. We also prove that this indeed computes the relation.

```
UTXO-step : Coin → UTxO × Coin → Tx → Maybe (UTxO × Coin)
UTXO-step minFee (utxo , fees) tx =
  if txins tx ⊆ᵇ dom utxo
    ∧ minFee ≤ᵇ txfee tx
    ∧ balance (txins tx ◃ utxo) ≡ᵇ (balance (outs tx) + txfee tx)
    then just ((txins tx ⋪ utxo) ∪ outs tx , fees + txfee tx)
    else nothing

UTXO-step-computes-UTXO :
    minFee ⊢ utxoState ⇀⦇ tx ,UTXO⦈ utxoState'
  ⇔ UTXO-step minFee utxoState tx ≡ just utxoState'
```

Figure 5: Computing the UTXO transition system

We prove this by considering both cases separately. Both cases follow easily by comparing the proof-carrying properties with the computational properties.