$$txb := \mathsf{txbody}\ tx \qquad \mathsf{txttl}\ txb \geq slot$$
$$\mathsf{txins}\ txb \neq \emptyset \qquad \mathsf{minfee}\ pp\ tx \leq \mathsf{txfee}\ txb \qquad \mathsf{txins}\ txb \subseteq \mathrm{dom}\ utxo$$
$$\mathsf{consumed}\ pp\ utxo\ txb = \mathsf{produced}\ pp\ poolParams\ txb$$

$$\begin{matrix} slot \\ pp \\ genDelegs \end{matrix} \vdash ppup \xrightarrow[\text{PPUP}]{\mathsf{txup}\ tx} ppup'$$

$$\forall(\_ \mapsto (\_,\ c)) \in \mathsf{txouts}\ txb, c \geq (\mathsf{minUTxOValue}\ pp)$$
$$\forall(\_ \mapsto (a,\ \_)) \in \mathsf{txouts}\ txb, a \in \mathsf{Addr}_{\mathsf{bootstrap}} \Rightarrow \mathsf{bootstrapAttrsSize}\ a \leq 64$$
$$\forall(\_ \mapsto (a,\ \_)) \in \mathsf{txouts}\ txb, \mathsf{netId}\ a = \mathsf{NetworkId}$$
$$\forall(a \mapsto \_) \in \mathsf{txwdrls}\ txb, \mathsf{netId}\ a = \mathsf{NetworkId}$$
$$\mathsf{txsize}\ tx \leq \mathsf{maxTxSize}\ pp$$

$$\text{UTxO-inductive}\ \frac{\begin{matrix} refunded := \mathsf{keyRefunds}\ pp\ txb \\ depositChange := \mathsf{totalDeposits}\ pp\ poolParams\ (\mathsf{txcerts}\ txb) - refunded \end{matrix}}{\begin{matrix} slot \\ pp \\ poolParams \\ genDelegs \end{matrix} \vdash \begin{pmatrix} utxo \\ deposited \\ fees \\ ppup \end{pmatrix} \xrightarrow[\text{UTXO}]{tx} \begin{pmatrix} (\mathbf{txins}\ txb \lhd utxo) \cup \mathbf{outs}\ txb \\ deposited + depositChange \\ fees + \mathbf{txfee}\ txb \\ ppup' \end{pmatrix}} \tag{4}$$

Figure 16: UTxO inference rules

- In the case of any input not being valid, there is a *BadInput* failure.
- In the case of the current slot being greater than the time-to-live of the current transaction, there is an *Expired* failure.
- In the case that the transaction is too large, there is a *MaxTxSize* failure.
- In the case of an empty input set, there is a *InputSetEmpty* failure, in order to prevent replay attacks.
- If the fees are smaller than the minimal transaction fees, there is a *FeeTooSmall* failure.
- If the transaction does not correctly conserve the balance, there is a *ValueNotConserved* failure.
- If the transaction creates any outputs with the wrong network ID, there is a *WrongNetwork* failure.
- If the transaction contains any withdrawals with the wrong network ID, there is a *WrongNetworkWithdrawal* failure.
- If the transaction creates an output below the allowed minimum value, there is a *OutputTooSmall* failure.
- If the transaction creates any boostrap outputs whose attributes have size bigger than 64, there is a *OutputBootAddrAttrsTooBig* failure.

8.2 Deposits and Refunds

Deposits are described in appendix B.2 of the delegation design document [SL-D1]. These deposit functions were used above in the UTxO transition, 8.1. Deposits are used for stake credential registration and pool registration certificates, which will be explained in Section 9. In particular, the function cwitness , which gets the certificate witness from a certificate, will be defined later. Figure 17 defines the deposit and refund functions.

- The function totalDeposits returns the total deposits that have to be made by a transaction. This calculation uses two protocol parameters, namely the key deposit value and the pool deposit value. Note that those certificates which are updates of stake pool parameters of already registered pool keys should not (and are, in fact, not allowed to) make a deposit.
- The function keyRefunds, calculates the total amount of returned deposits from stake key deregistration certificates.

  Note that keyRefunds uses the *current* protocol parameters. This means that any deposits made prior to a change in the deposit values will be refunded with the current value, not the one originally paid.

  The protocol parameters are not expected to change often and using the current ones for the calculation is a deliberate simplification choice, which does not introduce any inconsistencies into the system rules or properties. In particular, the general accounting property is not violated.

$$\mathsf{totalDeposits} \in \mathsf{PParams} \to (\mathsf{KeyHash}_{pool} \mapsto \mathsf{PoolParam}) \to \mathsf{DCert}^* \to \mathsf{Coin}$$ total deposits for a tx

$$\mathsf{totalDeposits}\ pp\ poolParams\ certs = (\mathsf{keyDeposit}\ pp) \cdot |certs \cap \mathsf{DCert}_{\mathsf{regkey}}| + (\mathsf{poolDeposit}\ pp) \cdot |\{\mathsf{cwitness}\ c \mid c \in newPools\}|$$

where

$$newPools = \{c \mid c \in certs \cap \mathsf{DCert}_{\mathsf{regpool}},\ \mathsf{cwitness}\ c \notin poolParams\}$$

$$\mathsf{keyRefunds} \in \mathsf{PParams} \to \mathsf{TxBody} \to \mathsf{Coin}$$ key refunds for a tx

$$\mathsf{keyRefunds}\ pp\ tx = (\mathsf{keyDeposit}\ pp) \cdot |\mathsf{txcerts}\ tx \cap \mathsf{DCert}_{\mathsf{deregkey}}|$$

Figure 17: Functions used in Deposits - Refunds

8.3 Witnesses

The purpose of witnessing is make sure that the intended action is authorized by the holder of the signing key, providing replay protection as a consequence. Replay prevention is an inherent property of UTxO type accounting since transaction IDs are unique and we require all transaction to consume at least one input.

A transaction is witnessed by a signature and a verification key corresponding to this signature. The witnesses, together with the transaction body, form a full transaction. Every witness in a transaction signs the transaction body. Moreover, the witnesses are represented as finite maps from verification keys to signatures, so that any key that is required to sign a transaction only provides a single witness. This means that, for example, a transaction which includes a delegation certificate and a reward withdrawal corresponding to the same stake credential still only includes one signature.

Figure 18 defines the function which gathers all the (hashes of) verification keys needed to witness a given transaction. This consists of:

- payment keys for outputs being spent
- stake credentials for reward withdrawals
- stake credentials for delegation certificates (except $\mathsf{DCert}_{\mathsf{mir}}$ and $\mathsf{DCert}_{\mathsf{regkey}}$)
- delegates of the genesis keys for any protocol parameter updates
- stake credentials for the pool owners in a pool registration certificate

The UTxOW transition system adds witnessing to the previous UTxO transition system. Figure 19 defines the type for this transition.

Figure 20 defines UTxOW transition. It has six predicates:

- Every signature in the transaction is a valid signature of the transaction body.
- The set of (hashes of) verification keys given by the transaction is a subset of the set of needed (hashes of) verification keys.
- Every multisignature script is valid.
- The set of scripts given by the transaction is equal to the set of required scripts.
- Any instantaneous reward certificates have quorum agreement from the genesis nodes delegates.
- Either the transaction metadata hash and the transaction metadata are both absent, or the hash present in the body is actually equal to the hash of the metadata.

If the predicates are satisfied, the state is transitioned by the UTxO transition rule.

The UTXOW rule has eight predicate failures:

- In the case of an incorrect witness, there is a *InvalidWitnesses* failure.
- In the case of a missing witness, there is a *MissingVKeyWitnesses* failure.
- In the case of missing scripts, there is a *MissingScriptWitnesses* failure.
- In the case of an invalid script, there is a *ScriptWitnessNotValidating* failure.
- In the case of a lack of quorum on an instantaneous reward certificate, there is a *MIRInsufficientGenesisSigs* failure.

$$\mathsf{propWits} \in \mathsf{Update} \to \mathsf{GenesisDelegation} \to \mathbb{P}\ \mathsf{KeyHash} \qquad \text{hashkeys for proposals}$$

$$\mathsf{propWits}\ (pup,\ \_)\ genDelegs =$$
$$\{kh \mid gkh \mapsto (kh,\ \_) \in (\mathrm{dom}\ pup \lhd genDelegs)\}$$

$$\mathsf{certWitsNeeded}\ \mathsf{Tx} \to \mathbb{P}\ \mathsf{Credential} \qquad \text{certificates with witnesses}$$

$$\mathsf{certWitsNeeded}\ tx =$$
$$\bigcup \{\mathsf{cwitness}\ c \mid c \in \mathsf{txcerts}\ tx \setminus (\mathsf{DCert}_{\mathsf{regkey}} \cup \mathsf{DCert}_{\mathsf{mir}})\}$$

$$\mathsf{witsVKeyNeeded} \in \mathsf{UTxO} \to \mathsf{Tx} \to (\mathsf{KeyHash}_{\mathsf{G}} \mapsto \mathsf{VKey}) \to \mathbb{P}\ \mathsf{KeyHash} \qquad \text{required key hashes}$$

$$\begin{aligned}
&\mathsf{witsVKeyNeeded}\ utxo\ tx\ genDelegs = \\
&\quad \{\mathsf{paymentHK}\ a \mid i \mapsto (a, _) \in utxo,\ i \in \mathsf{txinsVKey}\ tx\} \\
&\cup\ \{\mathsf{stakeCred_r}\ a \mid a \mapsto _ \in \mathsf{Addr}^{\mathsf{vkey}}_{\mathsf{rwd}} \lhd \mathsf{txwdrls}\ tx\} \\
&\cup\ (\mathsf{Addr}^{\mathsf{vkey}} \cap \mathsf{certWitsNeeded}\ tx) \\
&\cup\ \mathsf{propWits}\ (\mathsf{txup}\ tx)\ genDelegs \\
&\cup \bigcup_{\substack{c \in \mathsf{txcerts}\ tx \\ c \in \mathsf{DCert}_{\mathsf{regpool}}}} \mathsf{poolOwners}\ c
\end{aligned}$$

$$\mathsf{scriptsNeeded} \in \mathsf{UTxO} \to \mathsf{Tx} \to \mathbb{P}\ \mathsf{ScriptHash} \qquad \text{required script hashes}$$

$$\begin{aligned}
&\mathsf{scriptsNeeded}\ utxo\ tx = \\
&\quad \{\mathsf{validatorHash}\ a \mid i \mapsto (a, _) \in utxo, \\
&\qquad i \in \mathsf{txinsScript}\ (\mathsf{txins}\ tx)\ utxo\} \\
&\cup\ \{\mathsf{stakeCred_r}\ a \mid a \in \mathrm{dom}(\mathsf{Addr}^{\mathsf{script}}_{\mathsf{rwd}} \lhd \mathsf{txwdrls}\ tx)\} \\
&\cup\ (\mathsf{Addr}^{\mathsf{script}} \cap \mathsf{certWitsNeeded}\ tx)
\end{aligned}$$

Figure 18: Functions used in witness rule

- In the case the transaction contains metadata, but the transaction body does not contain a metadata hash, there is a *MissingTxBodyMetadataHash* failure.
- In the case that the transaction body contains a metadata hash, but there is no metadata outside the body, there is a *MissingTxMetadata* failure.
- In the case that the metadata hash in the transaction body is not equal to the hash of the metadata outside the body, there is a *ConflictingMetadataHash* failure.

UTxO with witness transitions

$$\_ \vdash \_ \xrightarrow[\text{UTXOW}]{} \_ \subseteq \mathbb{P}\ (\mathsf{UTxOEnv} \times \mathsf{UTxOState} \times \mathsf{Tx} \times \mathsf{UTxOState})$$

Figure 19: UTxO with witness transition-system types

$$(utxo, \_, \_, \_) := utxoSt$$
$$witsKeyHashes := \{\mathsf{hashKey}\ vk | vk \in \mathrm{dom}(\mathsf{txwitsVKey}\ tx)\}$$

$$\forall hs \mapsto validator \in \mathsf{txwitsScript}\ tx,$$
$$\mathsf{hashScript}\ validator = hs \wedge \mathsf{validateScript}\ validator\ tx$$

$$\mathsf{scriptsNeeded}\ utxo\ tx = \mathrm{dom}(\mathsf{txwitsScript}\ tx)$$

$$txbodyHash := \mathsf{hash}\ (\mathsf{txbody}\ tx)$$
$$\forall vk \mapsto \sigma \in \mathsf{txwitsVKey}\ tx, \mathcal{V}_{vk}[\![txbodyHash]\!]_\sigma$$
$$\mathsf{witsVKeyNeeded}\ utxo\ tx\ genDelegs \subseteq witsKeyHashes$$

$$genSig := \{genDelegate \mid (genDelegate, \_) \in \mathrm{range}\ genDelegs\} \cap witsKeyHashes$$
$$\{c \in \mathsf{txcerts}\ tx \cap \mathsf{DCert_{mir}}\} \neq \emptyset \implies |genSig| \geq \mathsf{Quorum}$$

$$mdh := \mathsf{txMDhash}\ tx \qquad md := \mathsf{txMD}\ tx$$
$$(mdh = \diamond \wedge md = \diamond) \vee (mdh = \mathsf{hashMD}\ md)$$

$$\text{UTxO-wit}\ \frac{\begin{array}{r} slot \\ pp \\ poolParams \\ genDelegs \end{array} \vdash utxoSt \xrightarrow[\text{UTXO}]{tx} utxoSt'}{\begin{array}{r} slot \\ pp \\ poolParams \\ genDelegs \end{array} \vdash utxoSt \xrightarrow[\text{UTXOW}]{tx} \boldsymbol{utxoSt'}} \quad (5)$$

Figure 20: UTxO with witnesses inference rules