6 Transactions

Transactions are defined in Figure 10. A transaction body, TxBody, is made up of eight 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 list of certificates, which will be explained in detail in Section 9.
- A transaction fee. This value will be added to the fee pot and eventually handed out as stake rewards.
- A time to live. A transaction will be deemed invalid if processed after this slot.
- A mapping of reward account withdrawals. The type Wdrl is a finite map that maps a reward address to the coin value to be withdrawn. The coin value must be equal to the full value contained in the account. Explicitly stating these values ensures that error messages can be precise about why a transaction is invalid. For reward calculation rules, see Section 11.1, and for the rule for collecting rewards, see Section 8.1.
- An optional update proposals for the protocol parameters. The update system will be explained in Section 7.
- An optional metadata hash.

A transaction, Tx, consists of:

- The transaction body.
- A triple of:
  - A finite map from payment verification keys to signatures.
  - A finite map containing scripts as values, with their hashes as their indexes.
  - Optional metadata.

Additionally, the UTxO type will be used by the ledger state to store all the unspent transaction outputs. It is a finite map from transaction inputs to transaction outputs that are available to be spent.

Finally, txid computes the transaction id of a given transaction. This function must produce a unique id for each unique transaction body.

Figure 11 shows the helper functions txinsVKey and txinsScript which partition the set of transaction inputs of the transaction into those that are locked with a private key and those that are locked via a script. It also defines validateScript, which validates the multisignature scripts.

Abstract types

$$\begin{array}{rclr}
\mathit{gkey} & \in & \mathsf{VKey}_G & \text{genesis public keys}\\
\mathit{gkh} & \in & \mathsf{KeyHash}_G & \text{genesis key hash}\\
\mathit{txid} & \in & \mathsf{TxId} & \text{transaction id}\\
m & \in & \mathsf{Metadatum} & \text{metadatum}\\
\mathit{mdh} & \in & \mathsf{MetadataHash} & \text{hash of transaction metadata}
\end{array}$$

Derived types

$$\begin{array}{rclclr}
(\mathit{txid}, \mathit{ix}) & \in & \mathsf{TxIn} & = & \mathsf{TxId} \times \mathsf{Ix} & \text{transaction input}\\
(\mathit{addr}, c) & \in & \mathsf{TxOut} & = & \mathsf{Addr} \times \mathsf{Coin} & \text{transaction output}\\
\mathit{utxo} & \in & \mathsf{UTxO} & = & \mathsf{TxIn} \mapsto \mathsf{TxOut} & \text{unspent tx outputs}\\
\mathit{wdrl} & \in & \mathsf{Wdrl} & = & \mathsf{Addr}_{\mathsf{rwd}} \mapsto \mathsf{Coin} & \text{reward withdrawal}\\
\mathit{md} & \in & \mathsf{Metadata} & = & \mathbb{N} \mapsto \mathsf{Metadatum} & \text{metadata}
\end{array}$$

Derived types (update system)

$$\begin{array}{rclclr}
\mathit{pup} & \in & \mathsf{ProposedPPUpdates} & = & \mathsf{KeyHash}_G \mapsto \mathsf{PParamsUpdate} & \text{proposed updates}\\
\mathit{up} & \in & \mathsf{Update} & = & \mathsf{ProposedPPUpdates} \times \mathsf{Epoch} & \text{update proposal}
\end{array}$$

Transaction Types

$$\begin{array}{rclcl}
\mathit{txbody} & \in & \mathsf{TxBody} & = & \begin{array}{l}\mathbb{P}\,\mathsf{TxIn} \times (\mathsf{Ix} \mapsto \mathsf{TxOut}) \times \mathsf{DCert}^{*} \times \mathsf{Coin} \times \mathsf{Slot} \times \mathsf{Wdrl}\\ \quad \times\, \mathsf{Update}^{?} \times \mathsf{MetadataHash}^{?}\end{array}\\
\mathit{wit} & \in & \mathsf{TxWitness} & = & (\mathsf{VKey} \mapsto \mathsf{Sig}) \times (\mathsf{ScriptHash} \mapsto \mathsf{Script})\\
\mathit{tx} & \in & \mathsf{Tx} & = & \mathsf{TxBody} \times \mathsf{TxWitness} \times \mathsf{Metadata}^{?}
\end{array}$$

Accessor Functions

$$\begin{array}{rclr}
\mathsf{txins} & \in & \mathsf{Tx} \to \mathbb{P}\,\mathsf{TxIn} & \text{transaction inputs}\\
\mathsf{txouts} & \in & \mathsf{Tx} \to (\mathsf{Ix} \mapsto \mathsf{TxOut}) & \text{transaction outputs}\\
\mathsf{txcerts} & \in & \mathsf{Tx} \to \mathsf{DCert}^{*} & \text{delegation certificates}\\
\mathsf{txfee} & \in & \mathsf{Tx} \to \mathsf{Coin} & \text{transaction fee}\\
\mathsf{txttl} & \in & \mathsf{Tx} \to \mathsf{Slot} & \text{time to live}\\
\mathsf{txwdrls} & \in & \mathsf{Tx} \to \mathsf{Wdrl} & \text{withdrawals}\\
\mathsf{txbody} & \in & \mathsf{Tx} \to \mathsf{TxBody} & \text{transaction body}\\
\mathsf{txwitsVKey} & \in & \mathsf{Tx} \to (\mathsf{VKey} \mapsto \mathsf{Sig}) & \text{VKey witnesses}\\
\mathsf{txwitsScript} & \in & \mathsf{Tx} \to (\mathsf{ScriptHash} \mapsto \mathsf{Script}) & \text{script witnesses}\\
\mathsf{txup} & \in & \mathsf{Tx} \to \mathsf{Update}^{?} & \text{protocol parameter update}\\
\mathsf{txMD} & \in & \mathsf{Tx} \to \mathsf{Metadata}^{?} & \text{metadata}\\
\mathsf{txMDhash} & \in & \mathsf{Tx} \to \mathsf{MetadataHash}^{?} & \text{metadata hash}
\end{array}$$

Abstract Functions

$$\begin{array}{rclr}
\mathsf{txid} & \in & \mathsf{TxBody} \to \mathsf{TxId} & \text{compute transaction id}\\
\mathsf{validateScript} & \in & \mathsf{Script} \to \mathsf{Tx} \to \mathsf{Bool} & \text{script interpreter}\\
\mathsf{hashMD} & \in & \mathsf{Metadata} \to \mathsf{MetadataHash} & \text{hash the metadata}\\
\mathsf{bootstrapAttrSize} & \in & \mathsf{Addr}_{\mathsf{bootstrap}} \to \mathbb{N} & \text{bootstrap attribute size}
\end{array}$$

Figure 10: Definitions used in the UTxO transition system

Helper Functions

$$\mathsf{txinsVKey} \in \mathbb{P}\ \mathsf{TxIn} \to \mathsf{UTxO} \to \mathbb{P}\ \mathsf{TxIn} \qquad \text{VKey Tx inputs}$$
$$\mathsf{txinsVKey}\ \mathit{txins}\ \mathit{utxo} = \mathit{txins} \cap \operatorname{dom}(\mathit{utxo} \rhd (\mathsf{Addr}^{\mathsf{vkey}} \times \mathit{Coin}))$$

$$\mathsf{txinsScript} \in \mathbb{P}\ \mathsf{TxIn} \to \mathsf{UTxO} \to \mathbb{P}\ \mathsf{TxIn} \qquad \text{Script Tx inputs}$$
$$\mathsf{txinsScript}\ \mathit{txins}\ \mathit{utxo} = \mathit{txins} \cap \operatorname{dom}(\mathit{utxo} \rhd (\mathsf{Addr}^{\mathsf{script}} \times \mathit{Coin}))$$

$$\mathsf{validateScript} \in \mathsf{Script} \to \mathsf{Tx} \to \mathsf{Bool} \qquad \text{validate script}$$
$$\mathsf{validateScript}\ \mathit{msig}\ \mathit{tx} = \begin{cases} \mathsf{evalMultiSigScript}\ \mathit{msig}\ \mathit{vhks} & \text{if } \mathit{msig} \in \mathsf{MSig} \\ \mathsf{False} & \text{otherwise} \end{cases}$$
$$\textbf{where}\ \ \mathit{vhks} := \{\mathsf{hashKey}\ \mathit{vk} | \mathit{vk} \in \operatorname{dom}(\mathsf{txwitsVKey}\ \mathit{tx})\}$$

Figure 11: Helper Functions for Transaction Inputs

7 Update Proposal Mechanism

The PPUP transition is responsible for the federated governance model in Shelley. The governance process includes a mechanism for core nodes to propose and vote on protocol parameter updates. In this chapter we outline rules for genesis keys *proposing* protocol parameter updates. For rules regarding the *adoption* of protocol parameter updates, see Section 11.7.

This chapter does not discuss authentication of update proposals. The signature for the keys in the proposal will be checked in the UTXOW transition, which checks all the necessary witnesses for a transaction, see Section8.3.

Genesis Key Delegations. The environment for the protocol parameter update transition contains the value *genDelegs*, which is a finite map indexed by genesis key hashes, and which maps to a pair consisting of a delegate key hash (corresponding to the cold key used for producing blocks) and a VRF key hash.

During the Byron era, the genesis nodes are all already delegated to some KeyHash, and these delegations are inherited through the Byron-Shelley transition (see Section 12.14). The VRF key hashes in this mapping will be new to the Shelley era.

The delegations mapping can be updated as described in Section 9, but there is no mechanism for them to un-delegate or for the keys to which they delegate to retire (unlike regular stake pools).

The types ProposedPPUpdates and Update were defined in Figure 10. The update proposal type Update is a pair of ProposedPPUpdates and Epoch. The epoch in the update specifies the epoch in which the proposal is valid. ProposedPPUpdates is a finite maps which is indexed by the hashes of the keys of entities proposing the given updates, $\mathsf{KeyHash}_G$. We use the abstract type $\mathsf{KeyHash}_G$ to represent hashes of genesis (public verification) keys, which have type VKey_G. Genesis keys are the keys belonging to the federated nodes running the Cardano system currently (also referred to as core nodes). The regular user verification keys are of a type VKey, distinct from the genesis key type, VKey_G. Similarly, the type hashes of these are distinct, KeyHash and $\mathsf{KeyHash}_G$ respectively.

Currently, updates can only be proposed and voted on by the owners of the genesis keys. The process of decentralization will result in the core nodes gradually giving up some of their privileges and responsibilities to the network, eventually give them *all* up. The update proposal mechanism will not be decentralized in the Shelley era, however. For more on the decentralization process, see Section 12.3.

7.1 Protocol Parameter Update Proposals

The transition type PPUP is for proposing updates to protocol parameters, see Figure 12 (for the corresponding rules, see Figure 13). The signal for this transition is an optional update.

Protocol updates for the current epoch are only allowed up until $(2 \cdot \mathsf{StabilityWindow})$-many slots before the end of the epoch. The reason for this involves how we safely predict hard forks. Changing the protocol version can result in a hard fork, and we would like an entire stability period between when we know that a hard fork will necessarily happen and when the current epoch ends. Protocol updates can still be submitted during the last $(2 \cdot \mathsf{StabilityWindow})$-many slots of the epoch, but they must be marked for the following epoch.

The transition PPUP has three rules:

- PP-Update-Empty : No new updates were proposed, do nothing.
- PP-Update-Current : Some new updates *up* were proposed for the current epoch, and the current slot is not too far into the epoch. Add these to the existing proposals using a right override. That is, if a genesis key has previously submitted an update proposal, replace it with its new proposal in *pup*.

- PP-Update-Future : Some new updates *up* were proposed for the next epoch, and the current slot is near the end of the epoch. Add these to the existing future proposals using a right override. That is, if a genesis key has previously submitted a future update proposal, replace it with its new proposal in *pup*.

  The future update proposals will become update proposals on the next epoch, provided they contain no proposals for a protocol version which cannot follow from the current protocol version. See the NEWPP transition in Figure 43, and the function updatePpup from Figure 42.

This rule has the following predicate failures:

1. In the case that the epoch number in the signal is not appropriate for the slot in the current epoch, there is a *PPUpdateWrongEpoch* failure.
2. In the case of *pup* being non-empty, if the check $\mathrm{dom}\,pup \subseteq \mathrm{dom}\,genDelegs$ fails, there is a *NonGenesisUpdate* failure as only genesis keys can be used in the protocol parameter update.
3. If a protocol parameter update in *pup* contains a proposal for a protocol version which cannot follow from the current protocol version, there is a *PVCannotFollow* failure. Note that pvCanFollow is defined in Figure 12.

Derived types

$$\mathsf{GenesisDelegation} \quad = \quad \mathsf{KeyHash_G} \mapsto (\mathsf{KeyHash} \times \mathsf{KeyHash}_{vrf}) \quad \text{genesis delegations}$$

Protocol Parameter Update environment

$$\mathsf{PPUpdateState} = \left(\begin{array}{rcll} pup & \in & \mathsf{ProposedPPUpdates} & \text{current proposals} \\ fpup & \in & \mathsf{ProposedPPUpdates} & \text{future proposals} \end{array}\right)$$

$$\mathsf{PPUpdateEnv} = \left(\begin{array}{rcll} slot & \in & \mathsf{Slot} & \text{current slot} \\ pp & \in & \mathsf{PParams} & \text{protocol parameters} \\ genDelegs & \in & \mathsf{GenesisDelegation} & \text{genesis key delegations} \end{array}\right)$$

Protocol Parameter Update transitions

$$\_ \vdash \_ \xrightarrow[\mathrm{PPUP}]{\_} \_ \subseteq \mathbb{P}\,(\mathsf{PPUpdateEnv} \times \mathsf{PPUpdateState} \times \mathsf{Update}^? \times \mathsf{PPUpdateState})$$

Helper Functions

$$\begin{array}{l} \mathsf{pvCanFollow} \in \mathsf{ProtVer} \to \mathsf{ProtVer} \to \mathsf{Bool} \\ \mathsf{pvCanFollow}\ (m,\ n)\ (m',\ n') = \\ \quad (m+1,0) = (m',n') \vee (m,n+1) = (m',n') \end{array}$$

Figure 12: Protocol Parameter Update Transition System Types