Formal Specification of the Cardano Ledger for the Babbage era

Andre Knispel
andre.knispel@iohk.io
Jared Corduan
jared.corduan@iohk.io

Abstract

This document presents the modifications to the Alonzo ledger specification (see Formal Methods Team, IOHK (2021)) for the Babbage era. The Babbage era introduces two main groups of changes.

The first group involves new ways of providing data to Plutus scripts. In particular, there is now support for reference inputs, inline datums, and reference scripts. Additionally, the Babbage era supports collateral outputs, which supports collateral outputs and asserting the exact collateral. The former helps with managing collateral for all wallets and the latter helps reduces the risk of using collateral inputs in hardware wallets.

The second group of changes involves the handling of block headers. We introduce a performance optimization, namely using a single VRF value for both the leader check and the epoch nonce contribution. We also remove the features introduced in the Shelley era which existed in order to smoothly transition from a federated environment into a decentralized environment (with respect to block production). In particular, there is no longer an overlay schedule or a mechanism for adding extra entropy to the epoch nonce.

List of Contributors

Alex Byaly, Nicholas Clarke, Duncan Coutts, Sebastien Guillemot, Philipp Kant, Jan Mazak, Michal Peyton Jones, Tim Sheard, Polina Vinogradova, Jamie Harper

Contents

List of Figures

1 Introduction

This specification describes the incremental changes from the Alonzo era of Cardano to the Babbage era. The main objective of this era is to make small adjustments in many places, usually to simplify the ledger or to include features that didn't make it into past eras. As part of this effort, we also make some changes to the notation used in these specifications, which should make them easier to understand and maintain.

Concretely, this specification makes the following changes:

- Add reference inputs to transactions
- Add inline datums in the UTxO
- Add reference scripts
- Add transaction fields for specifying and returning collateral
- Remove the protocol parameters d and $\mathit{extraEntropy}$
- Remove the overlay schedule
- Block headers to only include a single VRF value and proof
- Remove the pre-filtering of unregistered stake credentials in the reward calculation

2 Notation

This specification features some changes to the notation used in previous specifications.

Maps and partial functions We use the notation $f : A \to_* B$ to denote a finitely supported partial function. If B is a monoid, f is a function such that $fa = 0$ for all but finitely many a. Otherwise it is a function $f : A \to B^?$ such that $fa = \diamond$ for all but finitely many a.

Map operations We use standard notation for restriction and corestriction of functions to operate on partial functions as well.

3 Transactions

Derived Types

$$\mathsf{TxOut} = \mathsf{Addr} \times \mathsf{Value} \times (\mathsf{Datum} \uplus \mathsf{DataHash})^{?} \times \mathsf{Script}^{?}$$

Transaction Types

$\mathsf{TxBody} =$	$\mathbb{P}\ \mathsf{TxIn}$	spendInputs	Inputs
	$\times\ \mathbb{P}\ \mathsf{TxIn}$	collInputs	Collateral inputs
	$\times\ \mathbb{P}\ \mathsf{TxIn}$	refInputs	Reference inputs
	$\times\ (\mathsf{Ix} \to_{*} \mathsf{TxOut})$	txouts	Outputs
	$\times\ \mathsf{TxOut}^{?}$	collRet	Collateral return output
	$\times\ \mathsf{Coin}^{?}$	txcoll	Total collateral
	$\times\ \mathsf{DCert}^{*}$	txcerts	Certificates
	$\times\ \mathsf{Value}$	mint	A minted value
	$\times\ \mathsf{Coin}$	txfee	Total fee
	$\times\ \mathsf{ValidityInterval}$	txvldt	Validity interval
	$\times\ \mathsf{Wdrl}$	txwdrls	Reward withdrawals
	$\times\ \mathsf{Update}^{?}$	txUpdates	Update proposals
	$\times\ \mathbb{P}\ \mathsf{KeyHash}$	reqSignerHashes	Hashes of VKeys passed to scripts
	$\times\ \mathsf{ScriptIntegrityHash}^{?}$	scriptIntegrityHash	Hash of script-related data
	$\times\ \mathsf{AuxiliaryDataHash}^{?}$	txADhash	Auxiliary data hash
	$\times\ \mathsf{Network}^{?}$	txnetworkid	Tx network ID

Figure 1: Definitions for transactions

We add a field refInputs to the transaction that specifies *reference inputs*. A reference input is not spent and does not require any witnessing to be included in a valid transaction. The only requirement is that is has to be present in the ledger state UTxO. There are no restrictions on which outputs can be used as a reference input. Reference inputs only affect the information that is passed to scripts by them being included in TxInfo. For consistency, we've renamed the regular and collateral inputs to spendInputs and collInputs respectively.

We add two fields to the transaction dealing with collateral. collRet specifies outputs that get created in case a transaction script fails. txcoll asserts the total amount of collateral that would get collected as fees. Specifying this field does not change how collateral is computed, but transactions whose collateral is different than the amount in txcoll will be invalid. This lets users write transactions whose collateral is evident by just looking at the transaction body instead of requiring information contained in the UTXO, which hardware wallets for example might not have access to.

We also add support for supplying a Datum and a Script directly in a TxOut instead of just its hash. The *inline* Datum has two main purposes:

- In case of a sufficiently small Datum, this is more efficient
- Used together with reference inputs, this allows for many transactions to use a Datum without repeating it every time, thus reducing fees and block size

The *inline* script is visible to Plutus scripts and the scripts can be used together with reference inputs to not have to provide scripts in the transaction itself.

This change requires the size calculation of outputs to be adjusted, to properly scale with the new additions. For simplicity and future-proofing, we now use the serialized size.

4 UTxO

Some of the functions related to scripts, datums and collateral need to be adjusted for the new features. Most of these adjustments are self-explanatory. Note that the new collOuts function generates a single output with an index $|\mathsf{txouts}\ txb|$. This is to avoid potential confusion for transactions spending that output. Note that TxId can only hold integers up to $2^{16}-1$. In case of an overflow, we let this number be $2^{16}-1$.

Functions

$$\mathsf{isTwoPhaseScriptAddress} : \mathsf{Tx} \to \mathsf{UTxO} \to \mathsf{Addr} \to \mathsf{Bool}$$

$$\mathsf{isTwoPhaseScriptAddress}\ tx\ utxo\ a = \begin{cases} \mathsf{True} & a \in \mathsf{Addr}^{\mathsf{script}} \land \mathsf{validatorHash}\ a \mapsto s \in \mathsf{txscripts}\ tx\ utxo \land s \in \mathsf{Script}^{\mathsf{ph2}} \\ \mathsf{False} & \text{otherwise} \end{cases}$$

$$\mathsf{collOuts} : \mathsf{TxBody} \to \mathsf{UTxO}$$

$$\mathsf{collOuts}\ txb = \begin{cases} \emptyset & \mathsf{collRet}\ txb = \Diamond \\ \{(\mathsf{txid}\ txb, |\mathsf{txouts}\ txb|) \mapsto \mathsf{collRet}\ txb\} & \text{otherwise} \end{cases}$$

$$\mathsf{collBalance} : \mathsf{TxBody} \to \mathsf{UTxO} \to \mathsf{Value}$$

$$\mathsf{collBalance}\ txb\ utxo = \mathsf{ubalance}\ (utxo|_{\mathsf{collInputs}\ txb}) - \mathsf{ubalance}\ (\mathsf{collOuts}\ txb)$$

$$\mathsf{feesOK} : \mathsf{PParams} \to \mathsf{Tx} \to \mathsf{UTxO} \to \mathsf{Bool}$$

$$\begin{aligned} &\mathsf{feesOK}\ pp\ tx\ utxo = \\ &\quad \mathsf{minfee}\ pp\ tx \leq \mathsf{txfee}\ tx \land (\mathsf{txrdmrs}\ tx \neq \Diamond \Rightarrow \\ &\qquad \forall (a, \_, \_, \_) \in \mathsf{range}\ (\mathsf{collInputs}\ tx \lhd utxo), a \in \mathsf{Addr}^{\mathsf{vkey}} \\ &\qquad \land\ \mathsf{adaOnly}\ balance \\ &\qquad \land\ balance \geq \mathsf{txfee}\ txb * \mathsf{collateralPercent}\ pp / 100 \\ &\qquad \land\ (\mathsf{txcoll}\ tx \neq \Diamond) \Rightarrow balance = \mathsf{txcoll}\ tx \\ &\qquad \land\ \mathsf{collInputs}\ tx \neq \emptyset) \\ &\quad \textbf{where} \\ &\qquad balance = \mathsf{collBalance}\ tx\ utxo \end{aligned}$$

Figure 2: Functions related to fees and collateral

In the UTXO rule, we switch from a manual estimation of the size consumed by UTxO entries to an estimation using the serialization. However, since the TxIn used as a key in the UTxO map is not part of the serialization, we need to account for it manually. By itself it is 40 bytes, but we add another 120 bytes of overhead for the in-memory representation of Haskell data.

To the UTXOW rule, in addition to the changes required by the new features, we add a check that all scripts and datums involved in the transaction are well-formed. We forbid transactions that try to execute PlutusV1 scripts and use the new features which can't be translated to PlutusV1's TxInfo (reference inputs, inline datums and reference scripts). We also forbid transactions that involve Byron addresses.

$$\mathsf{getDatum} : \mathsf{Tx} \to \mathsf{UTxO} \to \mathsf{ScriptPurpose} \to \mathsf{Datum}^{*}$$

$$\mathsf{getDatum}\ tx\ utxo\ sp = \begin{cases} [d] & sp \in \mathsf{TxIn}, (\_, \_, h, \_) \in utxo\ sp,\ d \in \mathsf{txdats}\ (\mathsf{txwits}\ tx)\ h \\ [d] & sp \in \mathsf{TxIn}, (\_, \_, d, \_) \in utxo\ sp,\ d \in \mathsf{Datum} \\ \epsilon & \text{otherwise} \end{cases}$$

$$\mathsf{refScripts} : \mathsf{Tx} \to \mathsf{UTxO} \to \mathsf{ScriptHash} \to_{*} \mathsf{Script}$$

$$\mathsf{refScripts}\ tx\ utxo = \{\mathsf{hash}\ s \mapsto s \mid (\_, \_, \_, s) \in utxo\ (\mathsf{spendInputs}\ tx \cup \mathsf{refInputs}\ tx)\}$$

$$\mathsf{txscripts} : \mathsf{Tx} \to \mathsf{UTxO} \to \mathsf{ScriptHash} \to_{*} \mathsf{Script}$$

$$\mathsf{txscripts}\ tx\ utxo = \mathsf{txwitscripts}\ tx \cup \mathsf{refScripts}\ tx\ utxo$$

$$\mathsf{allOuts} : \mathsf{Tx} \to \mathbb{P}\ \mathsf{TxOut}$$

$$\mathsf{allOuts}\ tx = \mathrm{range}\,\mathsf{txouts}\ tx \cup \mathsf{collRet}\ tx$$

$$\mathsf{languages} : \mathsf{Tx} \to \mathsf{UTxO} \to \mathbb{P}\ \mathsf{Language}$$

$$\mathsf{languages}\ tx\ utxo = \{\mathsf{language}\ s \mid s \in \mathrm{range}(\mathsf{txscripts}\ tx\ utxo) \cap \mathsf{Script}^{\mathsf{ph2}}\}$$

$$\mathsf{allowedLanguages} : \mathsf{Tx} \to \mathsf{UTxO} \to \mathbb{P}\ \mathsf{Language}$$

$$\mathsf{allowedLanguages}\ tx\ utxo =$$

$$\begin{cases} \emptyset & \text{if } \exists (a, \_, \_, \_) \in os, a \in \mathsf{Addr}_{\mathsf{bootstrap}} \\ \{\mathsf{PlutusV2}\} & \text{if } \exists (\_, \_, d, s) \in os, d \in \mathsf{Datum} \lor s \neq \Diamond \lor \mathsf{refInputs}\ tx \neq \emptyset \\ \{\mathsf{PlutusV1}, \mathsf{PlutusV2}\} & \text{otherwise} \end{cases}$$

$$\textbf{where}\ \ os = \mathrm{range}\,\mathsf{txouts}\ tx \cup utxo\ (\mathsf{spendInputs}\ tx \cup \mathsf{refInputs}\ tx)$$

Figure 3: Functions related to scripts

$$\text{Scripts-Yes}\;\dfrac{\begin{array}{c} txb := \mathsf{txbody}\ tx \qquad sLst := \mathsf{collectTwoPhaseScriptInputs}\ pp\ tx\ utxo \\[1ex] \begin{array}{r} slot \\ pp \\ genDelegs \end{array} \vdash pup \xrightarrow[\text{PPUP}]{\mathsf{txup}\ tx} pup' \\[1ex] \begin{array}{r} refunded := \mathsf{keyRefunds}\ pp\ txb \\ depositChange := \mathsf{totalDeposits}\ pp\ poolParams\ (\mathsf{txcerts}\ txb)\ -\ refunded \end{array} \\[1ex] \mathsf{isValid}\ tx = \mathsf{evalScripts}\ tx\ sLst = \mathsf{True} \end{array}}{\begin{array}{l} slot \\ pp \\ poolParams \\ genDelegs \end{array} \vdash \left(\begin{array}{r} utxo \\ deposits \\ fees \\ pup \end{array}\right) \xrightarrow[\text{UTXOS}]{tx} \left(\begin{array}{r} (\mathbf{spendInputs}\ txb \lhd utxo) \cup \mathbf{outs}\ txb \\ deposits + depositChange \\ fees + \mathbf{txfee}\ txb \\ pup' \end{array}\right)} \tag{1}$$

$$\text{Scripts-No}\;\dfrac{\begin{array}{c} txb := \mathsf{txbody}\ tx \qquad sLst := \mathsf{collectTwoPhaseScriptInputs}\ pp\ tx\ utxo \\ collateralFees := \mathsf{valueToCoin}\ (\mathsf{collBalance}\ txb\ utxo) \\[1ex] \mathsf{isValid}\ tx = \mathsf{evalScripts}\ tx\ sLst = \mathsf{False} \end{array}}{\begin{array}{l} slot \\ pp \\ poolParams \\ genDelegs \end{array} \vdash \left(\begin{array}{r} utxo \\ deposits \\ fees \\ pup \end{array}\right) \xrightarrow[\text{UTXOS}]{tx} \left(\begin{array}{r} (\mathbf{collInputs}\ txb \lhd utxo) \cup \mathbf{collOuts}\ txb \\ deposits \\ fees + collateralFees \\ pup \end{array}\right)} \tag{2}$$

Figure 4: State update rules

$$
\text{UTxO-inductive}\ \dfrac{\begin{array}{c}
txb := \mathsf{txbody}\ tx \qquad \mathsf{ininterval}\ slot\ (\mathsf{txvldt}\ txb) \qquad (\_, i_f) := \mathsf{txvldt}\ tx \\[1ex]
\Diamond \notin \{\mathsf{txrdmrs}\ tx, i_f\} \Rightarrow \mathsf{epochInfoSlotToUTCTime}\ \mathsf{EI}\ \mathsf{SysSt}\ i_f \neq \Diamond \\
\mathsf{spendInputs}\ txb \neq \emptyset \qquad \mathsf{feesOK}\ pp\ tx\ utxo \\
\mathsf{spendInputs}\ txb \cup \mathsf{collInputs}\ txb \cup \mathsf{refInputs}\ tx \subseteq \mathrm{dom}\ utxo \\
\mathsf{consumed}\ pp\ utxo\ txb = \mathsf{produced}\ pp\ poolParams\ txb \\[1ex]
\mathsf{adaID} \notin \mathrm{supp}\ \mathsf{mint}\ tx \\[1ex]
\forall txout \in \mathsf{allOuts}\ txb, \\
\mathsf{getValue}\ txout \geq \mathsf{inject}\ (\ (\mathsf{serSize}\ txout + 160) * \mathsf{coinsPerUTxOByte}\ pp\ ) \\[1ex]
\forall txout \in \mathsf{allOuts}\ txb, \\
\mathsf{serSize}\ (\mathsf{getValue}\ txout) \leq \mathsf{maxValSize}\ pp \\[1ex]
\forall (\_ \mapsto (a, \_, \_, \_)) \in \mathsf{allOuts}\ txb, a \in \mathsf{Addr}_{\mathsf{bootstrap}} \Rightarrow \mathsf{bootstrapAttrsSize}\ a \leq 64 \\
\forall (\_ \mapsto (a, \_, \_, \_)) \in \mathsf{allOuts}\ txb, \mathsf{netId}\ a = \mathsf{NetworkId} \\
\forall (a \mapsto \_) \in \mathsf{txwdrls}\ txb, \mathsf{netId}\ a = \mathsf{NetworkId} \\
(\mathsf{txnetworkid}\ txb = \mathsf{NetworkId}) \vee (\mathsf{txnetworkid}\ txb = \Diamond) \\[1ex]
\mathsf{txsize}\ tx \leq \mathsf{maxTxSize}\ pp \\[1ex]
\mathsf{totExunits}\ tx \leq \mathsf{maxTxExUnits}\ pp \qquad \|\mathsf{collInputs}\ tx\| \leq \mathsf{maxCollateralInputs}\ pp \\
\begin{array}{c} slot \\ pp \\ poolParams \\ genDelegs \end{array} \vdash \left(\begin{array}{r} utxo \\ deposits \\ fees \\ pup \end{array}\right) \xrightarrow[\text{UTXOS}]{tx} \left(\begin{array}{r} utxo' \\ deposits' \\ fees' \\ pup' \end{array}\right)
\end{array}}{\begin{array}{l} slot \\ pp \\ poolParams \\ genDelegs \end{array} \vdash \left(\begin{array}{r} utxo \\ deposits \\ fees \\ pup \end{array}\right) \xrightarrow[\text{UTXO}]{tx} \left(\begin{array}{r} \boldsymbol{utxo'} \\ \boldsymbol{deposits'} \\ \boldsymbol{fees'} \\ \boldsymbol{pup'} \end{array}\right)} \tag{3}
$$

Figure 5: UTxO inference rules

$$
\begin{array}{c}
txb := \mathsf{txbody}\ tx \qquad\qquad\qquad\qquad txw := \mathsf{txwits}\ tx \\
(utxo, \_, \_, \_) := utxoSt \\
witsKeyHashes := \{\mathsf{hashKey}\ vk | vk \in \mathrm{dom}(\mathsf{txwitsVKey}\ txw)\} \\
inputHashes := \left\{ h \;\middle|\; \begin{array}{l} (a, \_, h, \_) \in \mathrm{range}(utxo|_{\mathsf{spendInputs}\ tx}) \\ \mathsf{isTwoPhaseScriptAddress}\ tx\ utxo\ a \end{array} \right\} - \mathsf{Datum} \\
neededHashes := \{h \mid (\_, h) \in \mathsf{scriptsNeeded}\ utxo\ txb\} \\
\forall s \in (\mathsf{txscripts}\ txw\ utxo\ neededHashes) \cap \mathsf{Script}^{\mathsf{ph1}}, \mathsf{validateScript}\ s\ tx \\[1ex]
neededHashes - \mathrm{dom}(\mathsf{refScripts}\ tx\ utxo) = \mathrm{dom}(\mathsf{txwitscripts}\ txw) \\[1ex]
inputHashes \subseteq_{\{h | (\_,\_,h,\_) \in \mathsf{allOuts}\ tx \cup\ utxo\ (\mathsf{refInputs}\ tx)\ \}} \mathrm{dom}(\mathsf{txdats}\ txw) \\[2ex]
\mathrm{dom}(\mathsf{txrdmrs}\ tx) = \left\{ \mathsf{rdptr}\ txb\ sp \;\middle|\; \begin{array}{l} (sp, h) \in \mathsf{scriptsNeeded}\ utxo\ txb \\ \mathsf{txscripts}\ txw\ utxo\ h \in \mathsf{Script}^{\mathsf{ph2}} \end{array} \right\} \\
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 \\[2ex]
genSig := \{\mathsf{hashKey}\ gkey | gkey \in \mathrm{dom}\ genDelegs\} \cap witsKeyHashes \\
\{c \in \mathsf{txcerts}\ txb\ \cap \mathsf{DCert}_{\mathsf{mir}}\} \neq \emptyset \implies |genSig| \geq \mathsf{Quorum} \\[2ex]
adh := \mathsf{txADhash}\ txb \qquad\qquad\qquad\qquad ad := \mathsf{auxiliaryData}\ tx \\
(adh = \Diamond \land ad = \Diamond) \lor (adh = \mathsf{hashAD}\ ad) \\[2ex]
\forall x \in \mathrm{range}(\mathsf{txdats}\ txw) \cup \mathrm{range}(\mathsf{txwitscripts}\ txw) \\
\cup \bigcup_{(\_,\_,d,s) \in\ \mathsf{allOuts}\ txb} \{s, d\} \cup \mathsf{scripts}\ (\mathsf{auxiliaryData}\ tx), \\
x \in \mathsf{Script} \cup \mathsf{Datum} \Rightarrow \mathsf{isWellFormed}\ x \\[2ex]
\mathsf{languages}\ tx\ utxo \subseteq \mathrm{dom}(\mathsf{costmdls}\ pp) \cap \mathsf{allowedLanguages}\ tx\ utxo \\
\mathsf{scriptIntegrityHash}\ txb = \mathsf{hashScriptIntegrity}\ pp\ (\mathsf{languages}\ txw\ utxo)\ (\mathsf{txrdmrs}\ txw)\ (\mathsf{txdats}\ txw) \\[2ex]
\begin{array}{r} slot \\ pp \\ poolParams \\ genDelegs \end{array} \vdash utxoSt \xrightarrow[\mathrm{UTXO}]{tx} utxoSt'
\end{array}
$$

$$
\text{UTxO-witG}\ \overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}
$$

$$
\begin{array}{r} slot \\ pp \\ poolParams \\ genDelegs \end{array} \vdash utxoSt \xrightarrow[\mathrm{UTXOW}]{tx} utxoSt' \tag{4}
$$

Figure 6: UTxO with witnesses inference rules for Tx

5 Removal of the Overlay Schedule

The overlay schedule was only used during the early days of the Shelley ledger, and can be safely removed. First, the protocol parameter d is removed, and any functions that use it are reduced to the case $d = 0$. The function mkApparentPerformance is reduced to one of its branches, and its first argument is dropped. It is only used in the definition of rewardOnePool, which needs to be adjusted accordingly.

Additionally, the block header body now contains a single VRF value to be used for both the leader check and the block nonce.

$$\begin{aligned}
&\mathsf{mkApparentPerformance} \in [0,\ 1] \to \mathbb{N} \to \mathbb{N} \to \mathbb{Q}\\
&\mathsf{mkApparentPerformance}\ \sigma\ n\ \overline{N} = \frac{\beta}{\sigma}\\
&\quad \textbf{where}\\
&\quad \beta = \frac{n}{\max(1, \overline{N})}
\end{aligned}$$

Figure 7: Function used in the Reward Calculation

The function createRUpd is adjusted by simplifying η.

Calculation to create a reward update

$$\begin{aligned}
&\mathsf{createRUpd} \in \mathbb{N} \to \mathsf{BlocksMade} \to \mathsf{EpochState} \to \mathsf{Coin} \to \mathsf{RewardUpdate}\\
&\mathsf{createRUpd}\ \mathit{slotsPerEpoch}\ b\ \mathit{es}\ \mathit{total} = (\Delta t_1, -\ \Delta r_1 + \Delta r_2,\ \mathit{rs},\ -\mathit{feeSS})\\
&\quad \textbf{where}\\
&\qquad \ldots\\
&\qquad \eta = \frac{\mathit{blocksMade}}{\lfloor \mathit{slotsPerEpoch} \cdot \mathsf{ActiveSlotCoeff} \rfloor}\\
&\qquad \ldots
\end{aligned}$$

Figure 8: Reward Update Creation

incrBlocks gets the same treatment as mkApparentPerformance. Its invocation in BBODY needs to be adjusted as well.

$$\begin{aligned}
&\mathsf{incrBlocks} \in \mathsf{KeyHash}_{pool} \to \mathsf{BlocksMade} \to \mathsf{BlocksMade}\\
&\mathsf{incrBlocks}\ hk\ b = \begin{cases} b \cup \{hk \mapsto 1\} & \text{if } hk \notin \operatorname{dom} b\\ b \underset{\rightarrow}{\cup} \{hk \mapsto n+1\} & \text{if } hk \mapsto n \in b \end{cases}
\end{aligned}$$

Finally, the PRTCL STS needs to be adjusted. To retire the OVERLAY STS, we inline the definition of its 'decentralized' case and drop all the unnecessary variables from its environment. It is invoked in CHAIN, which needs to be adjusted accordingly.

As there is now only a single VRF check, slight modifications are needed for the definition of the block header body BHBody type and the function vrfChecks. The Shelley era accessor functions bleader and bnonce are replaced with new functions which make use of the VRF range extension as described in Badertscher et al. (2022)[4.1], to re-use the single VRF value.

Block Header Body

$$\mathsf{BHBody} = \left(\begin{array}{rcll}
\mathit{prev} & \in & \mathsf{HashHeader}^? & \text{hash of previous block header} \\
\mathit{vk} & \in & \mathsf{VKey} & \text{block issuer} \\
\mathit{vrfVk} & \in & \mathsf{VKey} & \text{VRF verification key} \\
\mathit{blockno} & \in & \mathsf{BlockNo} & \text{block number} \\
\mathit{slot} & \in & \mathsf{Slot} & \text{block slot} \\
\mathit{vrfRes} & \in & \mathsf{Seed} & \text{VRF result value} \\
\mathit{prf} & \in & \mathsf{Proof} & \text{vrf proof} \\
\mathit{bsize} & \in & \mathbb{N} & \text{size of the block body} \\
\mathit{bhash} & \in & \mathsf{HashBBody} & \text{block body hash} \\
\mathit{oc} & \in & \mathsf{OCert} & \text{operational certificate} \\
\mathit{pv} & \in & \mathsf{ProtVer} & \text{protocol version}
\end{array}\right)$$

New Accessor Function

$$\begin{array}{rcl}
\mathsf{bVrfRes} & \in & \mathsf{BHBody} \to \mathsf{Seed} \\
\mathsf{bVrfProof} & \in & \mathsf{BHBody} \to \mathsf{Proof}
\end{array}$$

New Helper Functions

$$\mathsf{bleader} \in \mathsf{BHBody} \to \mathsf{Seed}$$
$$\mathsf{bleader}\ (\mathit{bhb}) = \mathsf{hash}\ (\text{“}L\text{”} \mid (\mathsf{bVrfRes}\ \mathit{bhb}))$$

$$\mathsf{bnonce} \in \mathsf{BHBody} \to \mathsf{Seed}$$
$$\mathsf{bnonce}\ (\mathit{bhb}) = \mathsf{hash}\ (\text{“}N\text{”} \mid (\mathsf{bVrfRes}\ \mathit{bhb}))$$

$$\mathsf{vrfChecks} \in \mathsf{Seed} \to \mathsf{BHBody} \to \mathsf{Bool}$$
$$\mathsf{vrfChecks}\ \eta_0\ \mathit{bhb} = \mathsf{verifyVrf}_{\mathsf{Seed}}\ \mathit{vrfK}\ (\mathsf{slotToSeed}\ \mathit{slot}\ \mathsf{XOR}\ \eta_0)\ (\mathit{value},\ \mathit{proof})$$

where

$$\begin{array}{l}
\mathit{slot} := \mathsf{bslot}\ \mathit{bhb} \\
\mathit{vrfK} := \mathsf{bvkvrf}\ \mathit{bhb} \\
\mathit{value} := \mathsf{bVrfRes}\ \mathit{bhb} \\
\mathit{proof} := \mathsf{bVrfProof}\ \mathit{bhb}
\end{array}$$

Figure 9: Block Definitions

Protocol environments

$$\mathsf{PrtclEnv} = \left(\begin{array}{rcll} pd & \in & \mathsf{PoolDistr} & \text{pool stake distribution} \\ \eta_0 & \in & \mathsf{Seed} & \text{epoch nonce} \end{array}\right)$$

Figure 10: Protocol transition-system types

$$bhb := \mathsf{bheader}\ bh \qquad \eta := \mathsf{bnonce}\ (\mathsf{bhbody}\ bhb)$$

$$\text{PRTCL}\ \frac{\begin{array}{c} \eta \vdash \begin{pmatrix} \eta_v \\ \eta_c \end{pmatrix} \xrightarrow[\text{UPDN}]{slot} \begin{pmatrix} \eta_v' \\ \eta_c' \end{pmatrix} \\ \vdash cs \xrightarrow[\text{OCERT}]{bh} cs' \\ \mathsf{praosVrfChecks}\ \eta_0\ pd\ \mathsf{ActiveSlotCoeff}\ bhb \end{array}}{\begin{array}{c} pd \\ \eta_0 \end{array} \vdash \begin{pmatrix} cs \\ \eta_v \\ \eta_c \end{pmatrix} \xrightarrow[\text{PRTCL}]{bh} \begin{pmatrix} cs' \\ \eta_v' \\ \eta_c' \end{pmatrix}} \qquad (5)$$

Figure 11: Protocol rules

6 Forgo Reward Calculation Prefilter

The reward calculation no longer filters out the unregistered stake credentials when creating a reward update. As in the Shelley era, though, they are still filtered on the epoch boundary when the reward update is applied. This addresses errata 17.2 in the Shelley ledger specification Formal Methods Team, IOHK (2019)[17.2]. The change consists of removing the line

$$\mathit{addrs}_{\mathit{rew}} \lhd \mathit{potentialRewards}$$

from the last line of the rewardOnePool function.

Calculation to reward a single stake pool

$$\begin{aligned}
&\mathsf{rewardOnePool} \in \mathsf{PParams} \to \mathsf{Coin} \to \mathbb{N} \to \mathbb{N} \to \mathsf{PoolParam}\\
&\quad \to \mathsf{Stake} \to \mathbb{Q} \to \mathbb{Q} \to \mathsf{Coin} \to (\mathsf{Addr}_{\mathsf{rwd}} \mapsto \mathsf{Coin})\\
&\mathsf{rewardOnePool}~\mathit{pp}~R~n~\overline{N}~\mathit{pool}~\mathit{stake}~\sigma~\sigma_a~\mathit{tot} = \mathit{rewards}\\
&\quad \textbf{where}\\
&\qquad \ldots\\
&\qquad \mathit{rewards} = \mathit{mRewards} \cup_{+} \{(\mathsf{poolRAcnt}~\mathit{pool}) \mapsto \mathit{lReward}\}
\end{aligned}$$

Figure 12: Reward Calculation Helper Function

A TxInfo Construction

The context of PlutusV2 needs to be adjusted to contain the new features. Additionally, the redeemers are provided to the context, but without the execution units budget.

Conversion Functions

$$\mathsf{toPlutusType}_{\mathsf{Script}} \in \mathsf{Script} \to \mathsf{P.ScriptHash}$$
$$\mathsf{toPlutusType}_{\mathsf{Script}}\ s = \mathsf{hash}\ s$$

$$\mathsf{toPlutusType}_{\mathsf{TxOut}} \in \mathsf{TxOut} \to \mathsf{P.TxOut}$$
$$\mathsf{toPlutusType}_{\mathsf{TxOut}}\ (a, v, d, s) = (a_P, v_P, d_P, s_P)$$

Figure 13: TxInfo Constituent Type Translation Functions

Ledger Functions

$$\mathsf{txInfo} : \mathsf{Language} \to \mathsf{PParams} \to \mathsf{EpochInfo} \to \mathsf{SystemStart} \to \mathsf{UTxO} \to \mathsf{Tx} \to \mathsf{TxInfo}$$

$$\begin{array}{l}
\mathsf{txInfo}\ \mathsf{PlutusV2}\ pp\ ei\ sysS\ utxo\ tx = \\
\quad (\{\ (txin_P, txout_P) \mid txin \in \mathsf{spendInputs}\ tx,\ txin \mapsto txout \in utxo\ \}, \\
\quad \{\ (txin_P, txout_P) \mid txin \in \mathsf{refInputs}\ tx,\ txin \mapsto txout \in utxo\ \}, \\
\quad \{\ tout_P \mid tout \in \mathsf{txouts}\ tx\ \}, \\
\quad (\mathsf{inject}\ (\mathsf{txfee}\ tx))_P, \\
\quad (\mathsf{mint}\ tx)_P, \\
\quad [\ c_P \mid c \in \mathsf{txcerts}\ tx\ ], \\
\quad \{\ (s_P,\ c_P) \mid s \mapsto c \in \mathsf{txwdrls}\ tx\ \}, \\
\quad \mathsf{transVITime}\ pp\ ei\ sysS\ (\mathsf{txvldt}\ tx), \\
\quad \{\ k_P \mid k \in \mathsf{dom}\,\mathsf{txwitsVKey}\ tx\ \}, \\
\quad \{(sp_P, d_P) \mid sp \mapsto (d, \_) \in \mathsf{indexedRdmrs}\ tx\}, \\
\quad \{\ (h_P,\ d_P) \mid h \mapsto d \in \mathsf{txdats}\ tx\ \}, \\
\quad (\mathsf{txid}\ tx)_P)
\end{array}$$

Figure 14: Transaction Summarization Functions

References

Christian Badertscher, Peter Gaži, Iñigo Querejeta-Azurmendi, and Alexander Russell. On uc-secure range extension and batch verification for ecvrf, 2022. URL https://iohk.io/en/research/library/papers/on-uc-secure-range-extension-and-batch-verification-for-ecvrf.

Formal Methods Team, IOHK. A Formal Specification of the Cardano Ledger, 2019. URL https://github.com/input-output-hk/cardano-ledger/releases/latest/download/shelley-ledger.pdf.

Formal Methods Team, IOHK. A Formal Specification of the Cardano Ledger with Plutus Integration, 2021. URL https://github.com/input-output-hk/cardano-ledger/tree/master/eras/alonzo/formal-spec/alonzo-ledger.tex.