If we assume that the *protocol parameters*, pp, are fixed[2], then we can provide convenience functions R_c and R_p for the *stake credential* and *stake pool* refunds, respectively:

$$\begin{aligned} R_c\ s_0\ s_1 &= \mathsf{refund}\ d_{val}\ d_{min}\ \lambda_d\ s_1 - s_0 \\ R_p\ s_0\ s_1 &= \mathsf{refund}\ p_{val}\ p_{min}\ \lambda_p\ s_1 - s_0 \end{aligned}$$

where d_{val}, d_{min}, λ_d, p_{val}, p_{min}, λ_p are the protocol parameter values from pp, and refund is defined in Figure 17. We let $\mathsf{DBE}(c,\ s)$ ("Deposits (precisely) Big Enough"), be the following property:

$$deposits = \left(\sum_{\_ \mapsto t \in C_{old}} R_c\ t\ \ell(s)\right) + |C_{new}| \cdot d_{val} + \left(\sum_{\_ \mapsto t \in P_{old}} R_p\ t\ \ell(s)\right) + |P_{new}| \cdot p_{val} \tag{DBE}$$

where

$$\begin{aligned} C_{old},\ C_{new} &= \mathsf{sep}\ stkCreds\ \ell(s) \\ P_{old},\ P_{new} &= \mathsf{sep}\ stpools\ \ell(s), \end{aligned}$$

for some slot, s, where pp, $stkCreds$, $stpools$ are in the corresponding chain state, c. In other words, $\mathsf{DBE}(c,\ s)$ asserts that the deposit pot is equal to the sum of the deposit refunds that were available at the previous epoch boundary, plus the sum of the initial deposit values for all the deposits from the current epoch.

Notice that for a chain state c and slot s, if the range of $stkCreds$ and $stpools$ contains only slots from the previous epoch, then $\mathsf{DBE}(c,\ s)$ is equivalent to

$$deposits = \mathsf{obligation}\ pp\ stkCreds\ stpools\ell(s) \tag{DEO}$$

where obligation is defined in Figure 34. It is generally true that $\mathsf{DBE}(c',\ s_i)$ holds after each subtransition of $s_i \vdash c_i \xrightarrow[\text{CHAIN}]{b_i} c_{i+1}$. However, this invariant can fail to hold after the DELEG transition, since this transition can add and remove stake credentials, and can also add stake pools, but the deposit pot is not adjusted accordingly until the next subtransiton of LEDGER, namely UTXO. The invariant can also fail to hold if the slot increases while the chain state remains the same. That is, if $\mathsf{DBE}(c_{i+1},\ s_i)$ holds, then $\mathsf{DBE}(c_{i+1},\ s_{i+1})$ can fail to hold if $\mathsf{epoch}\ s_i < \mathsf{epoch}\ s_{i+1}$, since the value of the deposit in the left hand side of equation DBE remains the same, but the refunded values become smaller[3]. Therefore, in this situation we can consider the slightly weaker constraint:

$$deposits \geq \mathsf{obligation}\ pp\ stkCreds\ stpools\ell(s) \tag{DGO}$$

The difference between the left and right hand sides of the inequality corresponds to the lovelace value in c_{i+1} that decays between s_i and s_{i+1}.

There are four sub-transitions where *deposits* is changed: SNAP (Figure 39), POOLREAP (Figure 41), NEWPP (Figure 43), UTXO (Figure 16). This ordering is also the order in which *deposits* is changed. Of these sub-transitions, only UTXO actually changes the value of *deposits* when s_i is in the same epoch as s_i. (We say that s_i *crosses the epoch boundary* if the precondition of Rule 27 in Figure 57 is met, namely if $\mathsf{epoch}\ s_i \geq e_\ell + 1$.) The proof then proceeds by induction on n, showing the following:

- Let c be the chain state after the SNAP transition in $s_i \vdash c_i \xrightarrow[\text{CHAIN}]{b_i} c_{i+1}$. If $\mathsf{DGO}(c_i,\ s_i)$, then $\mathsf{DBE}(c,\ s_i)$ holds.
- POOLREAP preserves DBE.

[2]Note that the protocol parameters can only change in the NEWPP transition.

[3]Note that if $\mathsf{epoch}\ s_i = \mathsf{epoch}\ s_{i+1}$, then $\mathsf{DBE}(c_{i+1},\ s_{i+1})$ is trivially true.

- NEWPP preserves DBE.
- The property for UTXO requires a bit of explanation. Let $nes \in \mathsf{NewEpochState}$ be the new epoch state in c_i. Note that the property DBE makes sense for values of NewEpochState since it contains all the relevant variables. Similarly, DBE also makes sense for values of $\mathsf{UTxOState} \times \mathsf{PParams}$. Let

$$gkeys \vdash nes \xrightarrow[\text{TICK}]{bh} nes'$$

be the first sub-transition of $s_i \vdash c_i \xrightarrow[\text{CHAIN}]{b_i} c_{i+1}$. If $\mathsf{DBE}(nes', s_i)$ holds, then $\mathsf{DBE}((us', pp), s_i)$ holds for every transaction tx in b_i, where:

$$env \vdash us \xrightarrow[\text{UTXO}]{tx} us',$$

is a sub-transition of $s_i \vdash c_i \xrightarrow[\text{CHAIN}]{b_i} c_{i+1}$, and pp is the protocol parameters in nes'.

Case SNAP. We must show that if c is the chain state after the SNAP transition in $s_i \vdash c_i \xrightarrow[\text{CHAIN}]{b_i} c_{i+1}$, and $\mathsf{DGO}(c_i, s_i)$ holds, then so does $\mathsf{DBE}(c, s_i)$. We can assume that s_i crosses the epoch boundary, since otherwise the SNAP transition will not occur. Since the SNAP transition only happens within the TICK transition on the epoch boundary, it follows that c_i does not contain any stake credentials or pools from the current epoch, and so DBE will be equivalent to DEO (the current epoch is epoch s_i). However, $\mathsf{DBE}(c, s_i)$ holds trivially, since it is determined from the obligation value.

Case POOLREAP. We must show that DBE is preserved. We again assume that s_i crosses the epoch boundary. The POOLREAP transition does the following:

1. leaves *stkCreds* unchanged,
2. removes *retired* from *stpools*,
3. subtracts *unclaimed* + *refunded* from *deposits*.

Notice that the domain of the *pr* is *retired*, and similarly the domain of the *rewardAcnts* is also *retired* since the domains of *stpools* and *poolParams* are the same. Therefore *retired* is the disjoint union of $\mathrm{dom}(refunds)$ and $\mathrm{dom}(mRefunds)$, so that

$$\begin{aligned} unclaimed + refunded &= \left(\sum_{\_ \mapsto t \in refunds} R_p\ t\ \ell(s) \right) + \left(\sum_{\_ \mapsto t \in mRefunds} R_p\ t\ \ell(s) \right) \\ &= \sum_{\_ \mapsto t \in rewardAcnts'} R_p\ t\ \ell(s) \\ &= \left(\sum_{\_ \mapsto t \in stpools} R_p\ t\ \ell(s) \right) - \left(\sum_{\_ \mapsto t \in retired \not\in stpools} R_p\ t\ \ell(s) \right) \end{aligned}$$

Therefore, it follows that if DEO holds before POOLREAP, then it also holds afterwards.

Case NEWPP. We must show that DBE is preserved. We again assume that s_i crosses the epoch boundary. In this transition *pp* can change, but *stkCreds*, *stpools*, and *deposits* do not change. As in the SNAP case, $\mathsf{DBE}(c, s_i)$ holds trivially, since it is set to the value that is determined by obligation.

Case UTXO. We assume that $\mathsf{DBE}(nes', s_i)$ holds, where nes' is the new epoch state after the TICK transition. We must show that DBE is preserved after each UTXO transition. The

DELEGS transition can result in values being added to or deleted from *stkCreds*, and added to *stpools*. Let A_s be the added stake credentials, D_s be the deleted credentials, and A_p be the added stake pools, where $stkCreds'$ is the stake credential mapping $stpools'$ is the stake pools, and $deposits'$ is the deposit pot after DELEGS. We have that

$$\begin{aligned} D_s &\subseteq stkCreds \cup A_s \\ stkCreds' &= (stkCreds \cup A_s) \setminus D_s \\ stpools' &= stpools \cup A_p \end{aligned}$$

The slots in the range of A_s will all be equal to s_i, but the slots in the range of D_s may either be from the current epoch or an earlier one, so we split them using sep:

$$(D_{s\_old},\ D_{s\_new}) = \mathsf{sep}\ D_s\ \ell(s_i)$$

We must then show that

$$deposits' = deposits + |A_s| \cdot d_{val} + |P_c| \cdot p_{val} - |D_{s\_new}| \cdot d_{val} - \left(\sum_{\_ \mapsto t \in D_{s\_old}} R_c\ t\ \ell(s_i) \right)$$

Looking at the UTXO transition in Figure 16,

$$deposits' = deposits + \mathsf{totalDeposits}\ pp\ stpools\ (\mathsf{txcerts}\ tx) - (refunded + decayed)$$

The function totalDeposits is defined in Figure 17 and it is clear that here it is equal to

$$|A_s| \cdot d_{val} + |P_c| \cdot p_{val.}$$

Recall that

$$\begin{aligned} refunded &= \mathsf{keyRefunds}\ pp\ stkCreds\ tx \\ decayed &= \mathsf{decayedTx}\ pp\ stkCreds\ \ tx \end{aligned}$$

where keyRefunds is defined in Figure 17. This iterates keyRefund from the same figure, which in turn just looks up the creation slot for a transaction and returns R_c. The function to calculate the value of decayed deposits, decayedTx, is defined in Figure **??**. This iterates decayedKey from the same figure. Therefore, to show that

$$|D_{s\_new}| \cdot d_{val} + \sum_{\_ \mapsto t \in D_{s\_old}} R_c\ t\ \ell(s_i) = refunded + decayed, \tag{47}$$

and thus complete the proof for the UTXO case, it suffices to show that for a given $c \mapsto s \in D_s$, the R_c value plus the decayedKey value that is associated with the stake credential c is equal to d_{val} if $\mathsf{epoch}\ (s) = \mathsf{epoch}\ (s_i)$, and is otherwise equal to $R_c\ s\ \ell(s_i)$. Looking at the definition of decayedKey, observe that if $\mathsf{epoch}\ (s) = \mathsf{epoch}\ (s_i)$ then $start = created$ and so the decayed value is $(R_c\ s\ s) - (R_c\ s\ s_i)$. However, $R_c\ s\ s = d_{val}$, so the refund plus the decayed value is $d_{val} - (R_c\ s\ s_i) + (R_c\ s\ s_i) = d_{val}$. Otherwise, if s is from a previous epoch, then $start = \ell(s_i)$, and so the decayed value is $(R_c\ s\ \ell(s_i)) - (R_c\ s\ s_i)$. The refund plus the decayed value is thus $(R_c\ s\ \ell(s_i)) - (R_c\ s\ s_i) + (R_c\ s\ s_i) = (R_c\ s\ \ell(s_i))$. Therefore, equation 47 holds, and consequently so also does $\mathsf{DBE}(c',\ s_i)$.

□

Values associated with the leader value calculations

$certNat$	$\in$	$\{n \mid n \in \mathbb{N}, n \in [0, 2^{512})\}$	Certified natural value from VRF
f	$\in$	$[0,1]$	Active slot coefficient
σ	$\in$	$[0,1]$	Stake proportion

16 Leader Value Calculation

This section details how we determine whether a node is entitled to lead (under the Praos protocol) given the output of its verifiable random function calculation.

16.1 Computing the leader value

The verifiable random function gives us a 64-byte random output. We interpret this as a natural number *certNat* in the range $[0, 2^{512})$.

16.2 Node eligibility

As per [DGKR17], a node is eligible to lead when its leader value $p < 1 - (1-f)^{\sigma}$. We have

$$\begin{aligned} & p < 1 - (1-f)^{\sigma} \\ \iff & \left(\frac{1}{1-p}\right) < \exp\left(-\sigma \cdot \ln\left(1-f\right)\right) \end{aligned}$$

The latter inequality can be efficiently computed through use of its Taylor expansion and error estimation to stop computing terms once we are certain that the result will be either above or below the target value.

We carry out all computations using fixed precision arithmetic (specifically, we use 34 decimal bits of precision, since this is enough to represent the fraction of a single lovelace.)

As such, we define the following:

$$\begin{aligned} p &= \frac{certNat}{2^{512}} \\ q &= 1 - p \\ c &= \ln\left(1-f\right) \end{aligned}$$

and define the function *checkLeaderVal* as follows:

$$\mathsf{checkLeaderVal}\ certNat\ \sigma\ f = \begin{cases} \mathsf{True}, & f = 1 \\ \frac{1}{q} < \exp\left(-\sigma \cdot c\right), & \text{otherwise} \end{cases}$$

17 Errata

We list issues found within the Shelley ledger which will be corrected in future eras.

17.1 Total stake calculation

As described in [SL-D1, 3.4.3], stake is sometimes considered relative to all the delegated stake, and sometimes relative to the totally supply less the amount held by the reserves. The former is called active stake, the latter total stake.

The createRUpd function from Figure 62 uses the current value of the reserve pot, named *reserves*, to calculate the total stake. It should, however, use the value of the reserve pot as it was during the previous epoch, since createRUpd is creating the rewards earned during the previous epoch (see the overview in Section 11.1).

17.2 Active stake registrations and the Reward Calculation

The reward calculation takes the set of active reward accounts as a parameter (the forth parameter of reward in Figure 48). It is only used at the end of the calculation for filtering out unregistered accounts. Therefore, if a stake credential is deregistered just before the reward calculation starts, it cannot reregister before the end of the epoch in order to get the rewards. The time within the epoch when the reward calculation starts should not make any difference. Note that this does not affect the earnings that stake pools make from their margin, since each pool's total stake is summed before filtering. The solution is to not filter by the current active reward accounts, since rewards for unregistered accounts go to the treasury already anyway.

17.3 Stability Windows

The constant *RandomnessStabilisationWindow* was intended to be used only for freezing the candidate nonce in the UPDN rule from Figure 60. This was chosen as a good value for both Ouroboros Praos and Ouroboros Genesis. The implementation mistakenly used the constant in the RUPD rule from Figure 62, and used StabilityWindow in for the UPDN transition.

The constant StabilityWindow is a good choice for the UPDN transition while we remain using Ouroboros Praos, but it will be changed before adopting Ouroboros Genesis in the consensus layer.

There is no problem with using *RandomnessStabilisationWindow* for the RUPD transition, since anything after StabilityWindow would work, but there is no reason to do so.

17.4 Reward aggregation

On any given epoch, a reward account can earn rewards by being a member of a stake pool, and also by being the registered reward account for a stake pool (to receive leader rewards). A reward account can be registered to receive leader rewards from multiple stake pools. It was intended that reward accounts receive the sum of all such rewards, but a mistake caused reward accounts to receive at most one of them.

In Figure 48, the value of *potentialRewards* in the rewardOnePool function should be computed using an aggregating union $\cup_+$, so that member rewards are not overridden by leader rewards. Similarly, the value of *rewards* in the reward function should be computed using an aggregating union so that leader rewards from multiple sources are aggregated.

This was corrected at the Allegra hard fork. There were sixty-four stake addresses that were affected, each of which was reimbursed for the exact amount lost using a MIR certificate. Four of the stake address were unregistered and had to be re-registered. The unregistered addresses were reimbursed in transaction