

9.3 Delegation Rules

The rules for registering and delegating stake credentials are given in [Figure 23](#). Note that section 5.2 of [SL-D1] describes how a wallet would help a user choose a stake pool, though these concerns are independent of the ledger rules.

- Stake credential registration is handled by [Equation \(6\)](#), since it contains the precondition that the certificate has type $\text{DCert}_{\text{regkey}}$. All the equations in DELEG and POOL follow this same pattern of matching on certificate type.

There is also a precondition on registration that the hashkey associated with the certificate witness of the certificate is not already found in the current reward accounts (which is the source of truth for which stake credentials are registered).

Registration causes the following state transformation:

- A reward account is created for this key, with a starting balance of zero.
- The certificate pointer is mapped to the new stake credential.
- Stake credential deregistration is handled by [Equation \(7\)](#). There is a precondition that the credential has been registered and that the reward balance is zero. Deregistration causes the following state transformation:
 - The key is removed from the collection of registered keys.
 - The reward account is removed.
 - The key is removed from the delegation relation.
 - The certificate pointer is removed.
- Stake credential delegation is handled by [Equation \(8\)](#). There is a precondition that the key has been registered. Delegation causes the following state transformation:
 - The delegation relation is updated so that the stake credential is delegated to the given stake pool. The use of union override here allows us to use the same rule to perform both an initial delegation and an update to an existing delegation.
- Genesis key delegation is handled by [Equation \(9\)](#). There is a precondition that the genesis key is already in the mapping *genDelegs*. Genesis delegation causes the following state transformation:
 - The future genesis delegation relation is updated with the new delegate to be adopted in *StabilityWindow*-many slots.
- Moving instantaneous rewards is handled by ?? and ??. There is a precondition that the current slot is early enough in the current epoch and that the available reserves or treasury are sufficient to pay for the instantaneous rewards.

The DELEG rule has ten possible predicate failures:

- In the case of a key registration certificate, if the staking credential is already registered, there is a *StakeKeyAlreadyRegistered* failure.
- In the case of a key deregistration certificate, if the key is not registered, there is a *StakeKeyNotRegistered* failure.
- In the case of a key deregistration certificate, if the associated reward account is non-zero, there is a *StakeKeyNonZeroAccountBalance* failure.

- In the case of a non-existing stake pool key in a delegation certificate, there is a *StakeDelegationImpossible* failure.
- In the case of a pool delegation certificate, there is a *WrongCertificateType* failure.
- In the case of a genesis key delegation certificate, if the genesis key is not in the domain of the genesis delegation mapping, there is a *GenesisKeyNotInMapping* failure.
- In the case of a genesis key delegation certificate, if the delegate key is in the range of the genesis delegation mapping, there is a *DuplicateGenesisDelegate* failure.
- In the case of insufficient reserves to pay the instantaneous rewards, there is a *Insufficient-ForInstantaneousRewards* failure.
- In the case that a MIR certificate is issued during the last StabilityWindow-many slots of the epoch, there is a *MIRCertificateTooLateinEpoch* failure.
- In the case of a genesis key delegation certificate, if the VRF key is in the range of the genesis delegation mapping, there is a *DuplicateGenesisVRF* failure.

$$\begin{array}{c}
\text{Deleg-Reg} \frac{c \in \text{DCert}_{\text{regkey}} \quad hk := \text{regCred } c \quad hk \notin \text{dom rewards}}{\text{slot } \vdash \begin{pmatrix} \text{rewards} \\ \text{delegations} \\ \text{ptrs} \\ f\text{GenDelegs} \\ \text{genDelegs} \\ i_{\text{rwd}} \end{pmatrix} \xrightarrow[\text{DELEG}]{c} \begin{pmatrix} \text{rewards} \cup \{hk \mapsto 0\} \\ \text{delegations} \\ \text{ptrs} \cup \{ptr \mapsto hk\} \\ f\text{GenDelegs} \\ \text{genDelegs} \\ i_{\text{rwd}} \end{pmatrix}} \quad (6)
\\[20pt]
\text{Deleg-Dereg} \frac{c \in \text{DCert}_{\text{deregkey}} \quad hk := \text{cwitness } c \quad hk \mapsto 0 \in \text{rewards}}{\text{slot } \vdash \begin{pmatrix} \text{rewards} \\ \text{delegations} \\ \text{ptrs} \\ f\text{GenDelegs} \\ \text{genDelegs} \\ i_{\text{rwd}} \end{pmatrix} \xrightarrow[\text{DELEG}]{c} \begin{pmatrix} \{hk\} \nrightarrow \text{rewards} \\ \{hk\} \nrightarrow \text{delegations} \\ \text{ptrs} \nrightarrow \{hk\} \\ f\text{GenDelegs} \\ \text{genDelegs} \\ i_{\text{rwd}} \end{pmatrix}} \quad (7)
\\[20pt]
\text{Deleg-Deleg} \frac{c \in \text{DCert}_{\text{delegate}} \quad hk := \text{cwitness } c \quad hk \in \text{dom rewards}}{\text{slot } \vdash \begin{pmatrix} \text{rewards} \\ \text{delegations} \\ \text{ptrs} \\ f\text{GenDelegs} \\ \text{genDelegs} \\ i_{\text{rwd}} \end{pmatrix} \xrightarrow[\text{DELEG}]{c} \begin{pmatrix} \text{rewards} \\ \text{delegations} \sqcup \{hk \mapsto \text{dpool } c\} \\ \text{ptrs} \\ f\text{GenDelegs} \\ \text{genDelegs} \\ i_{\text{rwd}} \end{pmatrix}} \quad (8)
\\[20pt]
\begin{array}{l}
c \in \text{DCert}_{\text{genesis}} \quad (gkh, vkh, vrf) := \text{genesisDeleg } c \\
s' := \text{slot} + \text{StabilityWindow} \quad gkh \in \text{dom genDelegs} \\
\text{cod} := \{(k, v) \mid (g \mapsto (k, v)) \in \text{genDelegs}, g \neq gkh\} \\
\text{fod} := \{(k, v) \mid ((_, g) \mapsto (k, v)) \in f\text{GenDelegs}, g \neq gkh\} \\
\text{currentOtherColdKeyHashes} := \{k \mid _ \mapsto (k, _) \in \text{cod}\} \\
\text{currentOtherVrfKeyHashes} := \{v \mid _ \mapsto (_, v) \in \text{cod}\} \\
\text{futureOtherColdKeyHashes} := \{k \mid _ \mapsto (k, _) \in \text{fod}\} \\
\text{futureOtherVrfKeyHashes} := \{v \mid _ \mapsto (_, v) \in \text{fod}\} \\
vkh \notin \text{currentOtherColdKeyHashes} \cup \text{futureOtherColdKeyHashes} \\
vrf \notin \text{currentOtherVrfKeyHashes} \cup \text{futureOtherVrfKeyHashes} \\
fdeleg := \{(s', gkh) \mapsto (vkh, vrf)\}
\end{array}
\\[20pt]
\text{Deleg-Gen} \frac{}{\text{slot } \vdash \begin{pmatrix} \text{rewards} \\ \text{delegations} \\ \text{ptrs} \\ f\text{GenDelegs} \\ \text{genDelegs} \\ i_{\text{rwd}} \end{pmatrix} \xrightarrow[\text{DELEG}]{c} \begin{pmatrix} \text{rewards} \\ \text{delegations} \\ \text{ptrs} \\ f\text{GenDelegs} \sqcup fdeleg \\ \text{genDelegs} \\ i_{\text{rwd}} \end{pmatrix}} \quad (9)
\end{array}$$

Figure 23: Delegation Inference Rules

$$\begin{array}{c}
c \in \text{DCert}_{\text{mir}} \\
\text{slot} < \text{firstSlot}((\text{epoch slot}) + 1) - \text{StabilityWindow} \\
(irR, irT) := i_{rwd} \quad (treasury, reserves) := acnt \\
(pot, irPot) := \begin{cases} (reserves, irR) & \text{mirPot } c = \text{ReservesMIR} \\ (treasury, irT) & \text{mirPot } c = \text{TreasuryMIR} \end{cases} \\
combined := (\text{mirTarget } c) \sqcup irPot \\
\sum_{\substack{\vdash \rightarrow val \in combined \\ \forall r \in \text{range } irPot, r \geq 0}} val \leq pot \\
i'_{rwd} := \begin{cases} (combined, irT) & \text{mirPot } c = \text{ReservesMIR} \\ (irR, combined) & \text{mirPot } c = \text{TreasuryMIR} \end{cases} \\
\text{Deleg-Mir} \frac{}{
\begin{array}{c}
\text{slot} \\
ptr \vdash \\
acnt
\end{array}
\left(\begin{array}{c}
rewards \\
delegations \\
ptrs \\
fGenDelegs \\
genDelegs \\
i_{rwd}
\end{array} \right) \xrightarrow[\text{DELEG}]{c} \left(\begin{array}{c}
rewards \\
delegations \\
ptrs \\
fGenDelegs \\
genDelegs \\
i'_{rwd}
\end{array} \right)
} \quad (10)
\end{array}$$

Figure 24: Move Instantaneous Rewards Inference Rule

9.4 Stake Pool Rules

The rules for updating the part of the ledger state defining the current stake pools are given in [Figure 25](#). The calculation of stake distribution is described in [Section 11.4](#).

In the pool rules, the stake pool is identified with the hashkey of the pool operator. For each rule, again, we first check that a given certificate c is of the correct type.

- Stake pool registration is handled by [Equation \(11\)](#). It is required that the pool not be currently registered. Registration causes the following state transformation:
 - The key is added to the set of registered stake pools.
 - The pool's parameters are stored.
- Stake pool parameter updates are handled by [Equation \(12\)](#). This rule, which also matches on the certificate type `DCertRegPool`, is distinguished from [Equation \(11\)](#) by the requirement that the pool be registered.

Unlike the initial stake pool registrations, the pool parameters will not change until the next epoch, after stake distribution snapshots are taken. This gives delegators an entire epoch to respond to changes in stake pool parameters. The staging is achieved by adding updates to the mapping $fPoolParams$, which will override $poolParam$ with new values in the EPOCH transition (see [Figure 45](#)).

This rule also ends stake pool retirements. Note that $poolParams$ is **not** updated. The registration creation slot does not change.

- Stake pool retirements are handled by [Equation \(13\)](#). Given a slot number $slot$, the application of this rule requires that the planned retirement epoch e stated in the certificate is in the future, i.e. after $epoch$ (the epoch of the current slot number in this context) and that it is no more than E_{max} epochs after the current one. It is also required that the pool be registered. Note that imposing the E_{max} constraint on the system is not strictly necessary. However, forcing stake pools to announce their retirement a shorter time in advance will curb the growth of the *retiring* list in the ledger state.

The pools scheduled for retirement must be removed from the *retiring* state variable at the end of the epoch they are scheduled to retire in. This non-signaled transition (triggered, instead, directly by a change of current slot number in the environment), along with all other transitions that take place at the epoch boundary, are described in [Section 11](#).

Retirement causes the following state transformation:

- The pool is marked to retire on the given epoch. If it was previously retiring, the retirement epoch is now updated.

The POOL rule has four predicate failures:

- In the case of a pool registration or re-registration certificate, if specified pool cost parameter is smaller than the value of the protocol parameter `minPoolCost`, there is a *StakePoolCostTooLow* failure.
- In the case of a pool retirement certificate, if the pool key is not in the domain of the stake pools mapping, there is a *StakePoolNotRegisteredOnKey* failure.
- In the case of a pool retirement certificate, if the retirement epoch is not between the current epoch and the relative maximal epoch from the current epoch, there is a *StakePoolRetirementWrongEpoch* failure.
- If the delegation certificate is not of one of the pool types, there is a *WrongCertificateType* failure.