we have that for all $i \in [1, n]$:

$$nes \vdash \tilde{s} \xrightarrow[\text{CHAINHEAD}]{H}{}^{*} s_h \wedge \vdash (nes,\ \tilde{s}) \xrightarrow[\text{CHAIN}]{[b_0 \ldots b_{i-1}]}{}^{*} s_{i-1} \implies nes' \vdash \tilde{s}_{i-1} \xrightarrow[\text{CHAINHEAD}]{h_i} s'_h$$

where $s_{i-1} = (nes',\ \tilde{s}_{i-1})$, t_E is the maximum slot number appearing in the blocks contained in E.

Property 15.2 states that if we validate a sequence of headers, we can validate their bodies independently and be sure that the blocks will pass the chain validation rule. To see this, given an environment e and initial state s, assume that a sequence of headers $H = [h_0, \ldots, h_n]$ corresponding to blocks in $E = [b_0, \ldots, b_n]$ is valid according to the chainhead transition system:

$$nes \vdash \tilde{s} \xrightarrow[\text{CHAINHEAD}]{H}{}^{*} \tilde{s'}$$

Assume the bodies of E are valid according to the bbody rules, but E is not valid according to the chain rule. Assume that there is a $b_j \in E$ such that it is **the first block** such that does not pass the chain validation. Then:

$$\vdash (nes,\ \tilde{s}) \xrightarrow[\text{CHAIN}]{[b_0, \ldots b_{j-1}]}{}^{*} s_j$$

But by Property 15.2 we know that

$$nes_j \vdash \tilde{s}_j \xrightarrow[\text{CHAINHEAD}]{h_j} \tilde{s}_{j+1}$$

which means that block b_j has valid headers, and this in turn means that the validation of b_j according to the chain rules must have failed because it contained an invalid block body. But this contradicts our assumption that the block bodies were valid.

Property 15.3 (Existence of roll back function)**.** There exists a function f such that for all chains

$$C = C_0; b; C_1$$

we have that if for all alternative chains C'_1, $|C'_1| \leq \frac{\mathsf{StabilityWindow}}{2}$, with corresponding headers H'_1

$$\vdash s_0 \xrightarrow[\text{CHAIN}]{C_0;b}{}^{*} s_1 \xrightarrow[\text{CHAIN}]{C_1}{}^{*} s_2 \wedge \vdash s_1 \xrightarrow[\text{CHAIN}]{C'_1}{}^{*} s'_1 \implies nes \vdash \tilde{s} \xrightarrow[\text{CHAINHEAD}]{H'_1}{}^{*} s_h$$

where $\mathsf{f}\ (\mathsf{bheader}\ b)\ s_2 = (nes,\ \tilde{s})$.

Property 15.3 expresses the fact the there is a function that allow us to recover the header-only state by rolling back at most k blocks, and use this state to validate the headers of an alternate chain. Note that this property is not inherent to the chain rules and can be trivially satisfied by any function that keeps track of the history of the intermediate chain states up to k blocks back. This property is stated here so that it can be used as a reference for the tests in the consensus layer, which uses the rules presented in this document.

15.2 Validity of a Ledger State

Many properties only make sense when applied to a valid ledger state. In informal terms, a valid ledger state l can only be reached when starting from an initial state l_0 (ledger in the genesis state) and only executing LEDGER state transition rules as specified in Section 10 which changes either the UTxO or the delegation state.

In Figure 81 $Genesis_{Id}$ marks the transaction identifier of the initial coin distribution, where $Genesis_{Out}$ represents the initial UTxO. It should be noted that no corresponding inputs exists, i.e., the transaction inputs are the empty set for the initial transaction. The function getUTxO extracts the UTxO from a UTxO state.

$$
\begin{array}{rr}
Genesis_{Id} \in & \mathsf{TxId} \\
Genesis_{Out} \in & \mathsf{TxOut} \\
Genesis_{UTxO} := & (Genesis_{Id}, 0) \mapsto Genesis_{Out} \\
\mathsf{ledgerState} \in & \left(\begin{array}{c} \mathsf{UTxOState} \\ \mathsf{DPState} \end{array} \right) \\
 & \\
\mathsf{getUTxO} \in & \mathsf{UTxOState} \to \mathsf{UTxO} \\
\mathsf{getUTxO} := & (utxo, \_, \_, \_) \mapsto utxo
\end{array}
$$

Figure 81: Definitions and Functions for Valid Ledger State

Definition 15.1 (Valid Ledger State).

$$
\forall l_0, \ldots, l_n \in \mathsf{LState}, lenv_0, \ldots, lenv_n \in \mathsf{LEnv}, l_0 = \left(\begin{array}{c} (Genesis_{UTxO}, \_, \_, \_) \\ \left(\begin{array}{c} \emptyset \\ \emptyset \end{array} \right) \end{array} \right)
$$
$$
\implies \forall 0 < i \leq n, (\exists tx_i \in \mathsf{Tx}, lenv_{i-1} \vdash l_{i-1} \xrightarrow[\text{LEDGER}]{tx_i} l_i) \implies \mathsf{validLedgerState}\ l_n
$$

Definition 15.1 defines a valid ledger state reachable from the genesis state via valid LEDGER STS transitions. This gives a constructive rule how to reach a valid ledger state.

15.3 Ledger Properties

The following properties state the desired features of updating a valid ledger state.

Property 15.4 (Preserve Balance).

$$
\forall l, l' \in \mathsf{LState} : \mathsf{validLedgerstate}\ l, l = (u, \_, \_, \_), l' = (u', \_, \_, \_)
$$
$$
\implies \forall tx \in \mathsf{Tx}, lenv \in \mathsf{LEnv}, lenv \vdash u \xrightarrow[\text{UTXOW}]{tx} u'
$$
$$
\implies \mathsf{destroyed}\ pc\ utxo\ stkCreds\ rewards\ tx = \mathsf{created}\ pc\ stPools\ tx
$$

Property 15.4 states that for each valid ledger l, if a transaction tx is added to the ledger via the state transition rule UTXOW to the new ledger state l', the balance of the UTxOs in l equals the balance of the UTxOs in l' in the sense that the amount of created value in l' equals the amount of destroyed value in l. This means that the total amount of value is left unchanged by a transaction.

Property 15.5 (Preserve Balance Restricted to TxIns in Balance of TxOuts).

$$
\forall l, l' \in \mathsf{ledgerState} : \mathsf{validLedgerstate}\ l, l = (u, \_, \_, \_), l' = (u', \_, \_, \_)
$$
$$
\implies \forall tx \in \mathsf{Tx}, lenv \in \mathsf{LEnv}, lenv \vdash u \xrightarrow[\text{UTXOW}]{tx} u'
$$
$$
\implies \mathsf{ubalance}(\mathsf{txins}\ tx \lhd \mathsf{getUTxO}\ u) = \mathsf{ubalance}(\mathsf{outs}\ tx) + \mathsf{txfee}\ tx + depositChange
$$

Property 15.5 states a slightly more detailed relation of the balances change. For ledgers l, l' and a transaction tx as above, the balance of the UTxOs of l restricted to those whose domain is in the set of transaction inputs of tx equals the balance of the transaction outputs of tx minus the transaction fees and the change in the deposit *depositChange* (cf. Fig. 16).

Property 15.6 (Preserve Outputs of Transaction).

$$\forall l, l' \in \mathsf{ledgerState} : \mathsf{validLedgerstate}\ l, l = (u, \_, \_, \_), l' = (u', \_, \_, \_)$$
$$\implies \forall tx \in \mathsf{Tx}, lenv \in \mathsf{LEnv}, lenv \vdash u \xrightarrow[\text{UTXOW}]{tx} u' \implies \forall out \in \mathsf{outs}\ tx, out \in \mathsf{getUTxO}\ u'$$

Property 15.6 states that for all ledger states l, l' and transaction tx as above, all output UTxOs of tx are in the UTxO set of l', i.e., they are now available as unspent transaction output.

Property 15.7 (Eliminate Inputs of Transaction).

$$\forall l, l' \in \mathsf{ledgerState} : \mathsf{validLedgerstate}\ l, l = (u, \_, \_, \_), l' = (u', \_, \_, \_)$$
$$\implies \forall tx \in \mathsf{Tx}, lenv \in \mathsf{LEnv}, lenv \vdash u \xrightarrow[\text{UTXOW}]{tx} u' \implies \forall in \in \mathsf{txins}\ tx, in \notin \mathsf{dom}(\mathsf{getUTxO}\ u')$$

Property 15.7 states that for all ledger states l, l' and transaction tx as above, all transaction inputs in of tx are not in the domain of the UTxO of l', i.e., these are no longer available to spend.

Property 15.8 (Completeness and Collision-Freeness of new Transaction Ids).

$$\forall l, l' \in \mathsf{ledgerState} : \mathsf{validLedgerstate}\ l, l = (u, \_, \_, \_), l' = (u', \_, \_, \_)$$
$$\implies \forall tx \in \mathsf{Tx}, lenv \in \mathsf{LEnv}, lenv \vdash u \xrightarrow[\text{UTXOW}]{tx} u'$$
$$\implies \forall ((txId', \_) \mapsto \_) \in \mathsf{outs}\ tx, ((txId, \_) \mapsto \_) \in \mathsf{getUTxO}\ u \implies txId' \neq txId$$

Property 15.8 states that for ledger states l, l' and a transaction tx as above, the UTxOs of l' contain all newly created UTxOs and the referred transaction id of each new UTxO is not used in the UTxO set of l.

Property 15.9 (Absence of Double-Spend).

$$\forall l_0, \ldots, l_n \in \mathsf{ledgerState}, l_0 = \begin{pmatrix} \{Genesis_{UTxO}\} \\ \begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix} \end{pmatrix} \wedge \mathsf{validLedgerState}\ l_n, l_i = (u_i, \_, \_, \_)$$
$$\implies \forall 0 < i \leq n, tx_i \in \mathsf{Tx}, lenv_i \in \mathsf{LEnv}, lenv_i \vdash u_{i-1} \xrightarrow[\text{LEDGER}]{tx_i} u_i \wedge \mathsf{validLedgerState}\ l_i$$
$$\implies \forall j < i, \mathsf{txins}\ tx_j \cap \mathsf{txins}\ tx_i = \emptyset$$

Property 15.9 states that for each valid ledger state l_n reachable from the genesis state, each transaction t_i does not share any input with any previous transaction t_j. This means that each output of a transition is spent at most once.

$$
\begin{aligned}
\mathsf{getStDelegs} &\in & \mathsf{DState} &\to \mathbb{P}\ \mathsf{Credential} \\
\mathsf{getStDelegs} &:= & (\mathit{stkCreds}, \_, \_, \_, \_, \_) &\to \mathit{stkCreds} \\
\\
\mathsf{getRewards} &\in & \mathsf{DState} &\to (\mathsf{Addr}_{\mathsf{rwd}} \mapsto \mathsf{Coin}) \\
\mathsf{getRewards} &:= & (\_, \mathit{rewards}, \_, \_, \_, \_) &\to \mathit{rewards} \\
\\
\mathsf{getDelegations} &\in & \mathsf{DState} &\to (\mathsf{Credential} \mapsto \mathsf{KeyHash}) \\
\mathsf{getDelegations} &:= & (\_, \_, \mathit{delegations}, \_, \_, \_) &\to \mathit{delegations} \\
\\
\mathsf{getStPools} &\in & \mathsf{LState} &\to (\mathsf{KeyHash} \mapsto \mathsf{DCert}_{\mathsf{regpool}}) \\
\mathsf{getStPools} &:= & (\_, (\_, (\mathit{stpools}, \_, \_, \_))) &\to \mathit{stpools} \\
\\
\mathsf{getRetiring} &\in & \mathsf{LState} &\to (\mathsf{KeyHash} \mapsto \mathsf{Epoch}) \\
\mathsf{getRetiring} &:= & (\_, (\_, (\_, \_, \mathit{retiring}, \_))) &\to \mathit{retiring}
\end{aligned}
$$

Figure 82: Definitions and Functions for Stake Delegation in Ledger States

15.4 Ledger State Properties for Delegation Transitions

Property 15.10 (Registered Staking Credential with Zero Rewards).

$$
\begin{aligned}
&\forall l, l' \in \mathsf{ledgerState} : \mathsf{validLedgerstate}\ l, l = (\_, ((d, \_), \_)), l' = (\_, ((d', \_), \_)), \mathit{dEnv} \in \mathsf{DEnv} \\
&\quad \implies \forall c \in \mathsf{DCert}_{\mathsf{regkey}}, \mathit{dEnv} \vdash d \xrightarrow[\text{DELEG}]{c} d' \implies \mathsf{cwitness}\ c = \mathit{hk} \\
&\implies \mathit{hk} \notin \mathsf{getStDelegs}\ d \implies \mathit{hk} \in \mathsf{getStDelegs}\ d' \wedge (\mathsf{getRewards}\ d')[\mathsf{addr}_{\mathsf{rwd}} \mathit{hk}] = 0
\end{aligned}
$$

Property 15.10 states that for each valid ledger state l, if a delegation transaction of type $\mathsf{DCert}_{\mathsf{regkey}}$ is executed, then in the resulting ledger state l', the set of staking credential of l' includes the credential hk associated with the key registration certificate and the associated reward is set to 0 in l'.

Property 15.11 (Deregistered Staking Credential).

$$
\begin{aligned}
&\forall l, l' \in \mathsf{ledgerState} : \mathsf{validLedgerstate}\ l, l = (\_, (d, \_)), l' = (\_, (d', \_)), \mathit{dEnv} \in \mathsf{DEnv} \\
&\quad \implies \forall c \in \mathsf{DCert}_{\mathsf{deregkey}}, \mathit{dEnv} \vdash d \xrightarrow[\text{DELEG}]{c} d' \implies \mathsf{cwitness}\ c = \mathit{hk} \\
&\implies \mathit{hk} \notin \mathsf{getStDelegs}\ d' \wedge \mathit{hk} \notin \left\{\mathsf{stakeCred}_{\mathsf{r}}\ \mathit{sc} \middle| \mathit{sc} \in \mathsf{dom}(\mathsf{getRewards}\ d')\right\} \\
&\qquad\qquad \wedge \mathit{hk} \notin \mathsf{dom}(\mathsf{getDelegations}\ d'))
\end{aligned}
$$

Property 15.11 states that for l, l' as above but with a delegation transition of type $\mathsf{DCert}_{\mathsf{deregkey}}$, the staking credential hk associated with the deregistration certificate is not in the set of staking credentials of l' and is not in the domain of either the rewards or the delegation map of l'.

Property 15.12 (Delegated Stake).

$$\begin{aligned}\forall l, l' \in \mathsf{ledgerState} : \mathsf{validLedgerstate}\ l, l = (\_, (d, \_)), l' = (\_, (d', \_)), dEnv \in \mathsf{DEnv} \\ \implies \forall c \in \mathsf{DCert_{delegate}}, dEnv \vdash d \xrightarrow[\text{DELEG}]{c} d' \implies \mathsf{cwitness}\ c = hk \\ \implies hk \in \mathsf{getStDelegs}\ d' \wedge (\mathsf{getDelegations}\ d')[hk] = \mathsf{dpool}\ c\end{aligned}$$

Property 15.12 states that for l, l' as above but with a delegation transition of type $\mathsf{DCert_{delegate}}$, the staking credential hk associated with the deregistration certificate is in the set of staking credentials of l and delegates to the staking pool associated with the delegation certificate in l'.

Property 15.13 (Genesis Keys are Always All Delegated).

$$\begin{aligned}\forall l, l' \in \mathsf{LState} : \mathsf{validLedgerstate}\ l, \\ \implies \forall \Gamma \in \mathsf{Tx}^*, env \in (\mathsf{Slot} \times \mathsf{PParams}), \\ env \vdash l \xrightarrow[\text{LEDGERS}]{\Gamma} l' \implies |genDelegs| = 7\end{aligned}$$

Property 15.13 states that all seven of the genesis keys are constantly all delegated after applying a list of transactions to a valid ledger state.

15.5 Ledger State Properties for Staking Pool Transitions

Property 15.14 (Registered Staking Pool).

$$\begin{aligned}\forall l, l' \in \mathsf{ledgerState} : \mathsf{validLedgerstate}\ l, l = (\_, (\_, p)), l' = (\_, (\_, p')), pEnv \in \mathsf{PEnv} \\ \implies \forall c \in \mathsf{DCert_{regpool}}, p \xrightarrow[\text{POOL}]{c} p' \implies \mathsf{cwitness}\ c = hk \\ \implies hk \in \mathsf{getStPools}\ p' \wedge hk \notin \mathsf{getRetiring}\ p'\end{aligned}$$

Property 15.14 states that for l, l' as above but with a delegation transition of type $\mathsf{DCert_{regpool}}$, the key hk is associated with the author of the pool registration certificate in *stpools* of l' and that hk is not in the set of retiring stake pools in l'.

Property 15.15 (Start Staking Pool Retirement).

$$\begin{aligned}\forall l, l' \in \mathsf{ledgerState}, cepoch \in \mathsf{Epoch} : \mathsf{validLedgerstate}\ l, l = (\_, (\_, p)), l' = (\_, (\_, p')), pEnv \in \mathsf{PEnv} \\ \implies \forall c \in \mathsf{DCert_{retirepool}}, pEnv \vdash p \xrightarrow[\text{POOL}]{c} p' \\ \implies e = \mathsf{retire}\ c \wedge cepoch < e < cepoch + \mathsf{E_{max}} \wedge \mathsf{cwitness}\ c = hk \\ \implies (\mathsf{getRetiring}\ p')[hk] = e \wedge hk \in \mathsf{dom}(\mathsf{getStPools}\ p) \wedge \mathsf{dom}(\mathsf{getStPools}\ p')\end{aligned}$$

Property 15.15 states that for l, l' as above but with a delegation transition of type $\mathsf{DCert_{retirepool}}$, the key hk is associated with the author of the pool registration certificate in *stpools* of l' and that hk is in the map of retiring staking pools of l' with retirement epoch e, as well as that hk is in the map of stake pools in l and l'.