

MIR Transitions

$$\vdash \_ \xrightarrow{\text{MIR}} \_ \subseteq \mathbb{P} (\text{EpochState} \times \text{EpochState})$$

**Figure 54:** MIR transition-system types

## 12.2 MIR Transition

The transition which moves the instantaneous rewards is MIR. Figure 54 defines the types for the transition. It has no environment or signal, and the state is EpochState.

Figure 55 defines the MIR state transition.

If the reserve and treasury pots are large enough to cover the sum of the corresponding instantaneous rewards, the reward accounts are increased by the appropriate amount and the two pots are decreased appropriately. In either case, if the pots are large enough or not, we reset both of the instantaneous reward mappings back to the empty mapping.

$$\begin{aligned}
 & (rewards, delegations, ptrs, fGenDelegs, genDelegs, i_{rwd}) := ds \\
 & (treasury, reserves) := acnt \quad (irReserves, irTreasury) := i_{rwd} \\
 & irwdR := \{addr_{rwd} \ hk \mapsto val \mid hk \mapsto val \in (\text{dom } rewards) \triangleleft irReserves\} \\
 & irwdT := \{addr_{rwd} \ hk \mapsto val \mid hk \mapsto val \in (\text{dom } rewards) \triangleleft irTreasury\} \\
 & totR := \sum_{\_ \mapsto v \in irwdR} v \quad totT := \sum_{\_ \mapsto v \in irwdT} v \\
 & enough := totR \leq reserves \wedge totT \leq treasury \\
 & acnt' := \begin{cases} (treasury - totT, reserves - totR) & \text{enough} \\ acnt & \text{otherwise} \end{cases} \\
 & rewards' := \begin{cases} rewards \cup_+ irwdR \cup_+ irwdT & \text{enough} \\ rewards & \text{otherwise} \end{cases} \\
 & ds' := (\textcolor{blue}{rewards}', delegations, ptrs, fGenDelegs, genDelegs, (\emptyset, \emptyset)) \\
 \text{MIR} \frac{}{\vdash \left( \begin{array}{c} acnt \\ ss \\ (us, (ds, ps)) \\ prevPP \\ pp \end{array} \right) \xrightarrow{\text{MIR}} \left( \begin{array}{c} \textcolor{blue}{acnt'} \\ ss \\ (us, (\textcolor{blue}{ds'}, ps)) \\ prevPP \\ pp \end{array} \right)} \quad (26)
 \end{aligned}$$

**Figure 55:** MIR rules

## 12.3 New Epoch Transition

For the transition to a new epoch (NEWEPOCH), the environment is given in Figure 56, it consists of

- The current slot.
- The set of genesis keys.

The new epoch state is given in Figure 56, it consists of

- The number of the last epoch.
- The information about produced blocks for each stake pool during the previous epoch.

*New Epoch states*

$$\text{NewEpochState} = \left( \begin{array}{ll} e_\ell \in \text{Epoch} & \text{last epoch} \\ b_{\text{prev}} \in \text{BlocksMade} & \text{blocks made last epoch} \\ b_{\text{cur}} \in \text{BlocksMade} & \text{blocks made this epoch} \\ es \in \text{EpochState} & \text{epoch state} \\ ru \in \text{RewardUpdate}^? & \text{reward update} \\ pd \in \text{PoolDistr} & \text{pool stake distribution} \end{array} \right)$$

*New Epoch Transitions*

$$\vdash - \xrightarrow[\text{NEW EPOCH}]{-} - \subseteq \mathbb{P} (\text{NewEpochState} \times \text{Epoch} \times \text{NewEpochState})$$

*Helper function*

$\text{calculatePoolDistr} \in \text{Snapshot} \rightarrow \text{PoolDistr}$

$\text{calculatePoolDistr} (\text{stake}, \text{delegs}, \text{poolParams}) =$

$$\left\{ hk_p \mapsto (\sigma, \text{poolIVRF } p) \mid \begin{array}{l} hk_p \mapsto \sigma \in sd \\ hk_p \mapsto p \in \text{poolParams} \end{array} \right\}$$

**where**

$$\text{total} = \sum_{\vdash \mapsto c \in \text{stake}} c$$

$$sd = \text{aggregate}_+ \left( \text{delegs}^{-1} \circ \left\{ \left( hk, \frac{c}{\text{total}} \right) \mid (hk, c) \in \text{stake} \right\} \right)$$

**Figure 56:** NewEpoch transition-system types

- The information about produced blocks for each stake pool during the current epoch.
- The old epoch state.
- An optional rewards update.
- The stake pool distribution of the epoch.

Figure 57 defines the new epoch state transition. It has three rules. The first rule describes the change in the case of  $e$  being equal to the next epoch  $e_\ell + 1$ . It also calls the MIR and EPOCH rules and checks that the reward update is net neutral with respect to the Ada in the system. This should always hold (by the definition of the createRupd function) and is present only for extra assurance and for help in proving that Ada is preserved by this transition. The second rule deals with the case when the epoch signal  $e$  is not one greater than the current epoch  $e_\ell$ . This rule does not change the state. The third rule is nearly the same as the first rule, only there is no reward update to apply. This third rule is defined for completeness, but in practice we hope that it is never used, since it only applies in the case that epoch  $e$  processed no blocks after the first StabilityWindow-many slots.

In the first case, the new epoch state is updated as follows:

- The epoch is set to the new epoch  $e$ .
- The mapping for the blocks produced by each stake pool for the previous epoch is set to the current such mapping.

- The mapping for the blocks produced by each stake pool for the current epoch is set to the empty map.
- The epoch state is updated with: first applying the rewards update  $ru$ , then calling the MIR transition, and finally by calling the EPOCH transition.
- The rewards update is set to  $\diamond$ .
- The new pool distribution  $pd'$  is calculated from the delegation map and stake allocation of the previous epoch.

$$\begin{array}{c}
 e = e_\ell + 1 \quad ru \neq \diamond \quad (\Delta t, \Delta r, rs, \Delta f) := ru \\
 \Delta t + \Delta r + \left( \sum_{\_ \mapsto v \in rs} v \right) + \Delta f = 0 \\
 es' := \text{applyRUupd } ru \ es \quad \vdash es' \xrightarrow[\text{MIR}]{} es'' \quad \vdash es'' \xrightarrow[\text{EPOCH}]{} es''' \\
 (acnt, ss, \_, \_, \_) := es''' \\
 (\_, pstake_{set}, \_, \_) := ss \\
 pd' := \text{calculatePoolDistr } pstake_{set} \\
 \hline
 \text{New-Epoch} \quad \vdash \begin{pmatrix} e_\ell \\ b_{prev} \\ b_{cur} \\ es \\ ru \\ pd \end{pmatrix} \xrightarrow[\text{NEW EPOCH}]{} \begin{pmatrix} e \\ b_{cur} \\ \emptyset \\ es''' \\ \diamond \\ pd' \end{pmatrix}
 \end{array} \tag{27}$$

$$\begin{array}{c}
 \text{Not-New-Epoch} \quad (e_\ell, \_, \_, \_, \_, \_) := nes \quad e \neq e_\ell + 1 \\
 \vdash nes \xrightarrow[\text{NEW EPOCH}]{} nes
 \end{array} \tag{28}$$

$$\begin{array}{c}
 (e_\ell, \_, \_, \_, ru, \_, \_) := nes \quad e = e_\ell + 1 \quad ru = \diamond \\
 \vdash es \xrightarrow[\text{MIR}]{} es'' \quad \vdash es'' \xrightarrow[\text{EPOCH}]{} es''' \\
 (acnt, ss, \_, \_, \_) := es''' \\
 (\_, pstake_{set}, \_, \_) := ss \\
 pd' := \text{calculatePoolDistr } pstake_{set} \\
 \hline
 \text{No-Reward-Update} \quad \vdash \begin{pmatrix} e_\ell \\ b_{prev} \\ b_{cur} \\ es \\ ru \\ pd \end{pmatrix} \xrightarrow[\text{NEW EPOCH}]{} \begin{pmatrix} e \\ b_{cur} \\ \emptyset \\ es''' \\ ru \\ pd' \end{pmatrix}
 \end{array} \tag{29}$$

Figure 57: New Epoch rules

*Tick Nonce environments*

$$\text{TickNonceEnv} = \left( \begin{array}{ll} pp \in \text{PParams} & \text{protocol parameters} \\ \eta_c \in \text{Seed} & \text{candidate nonce} \\ \eta_{ph} \in \text{Seed} & \text{previous header hash as nonce} \end{array} \right)$$

*Tick Nonce states*

$$\text{TickNonceState} = \left( \begin{array}{ll} \eta_0 \in \text{Seed} & \text{epoch nonce} \\ \eta_h \in \text{Seed} & \text{seed generated from hash of previous epoch's last block header} \end{array} \right)$$

## 12.4 Tick Nonce Transition

The Tick Nonce Transition is responsible for updating the epoch nonce and the previous epoch's hash nonce at the start of an epoch. Its environment is shown in Figure 12.4 and consists of the protocol parameters  $pp$ , the candidate nonce  $\eta_c$  and the previous epoch's last block header hash as a nonce. Its state consists of the epoch nonce  $\eta_0$  and the previous epoch's last block header hash nonce.

The signal to the transition rule TICKN is a marker indicating whether we are in a new epoch. If we are in a new epoch, we update the epoch nonce and the previous hash. Otherwise, we do nothing.

$$\begin{array}{c} \text{Not-New-Epoch} \frac{}{} \\ \frac{pp \quad \eta_c \quad \eta_{ph} \vdash \left( \begin{array}{c} \eta_0 \\ \eta_h \end{array} \right) \xrightarrow[\text{TICKN}]{\text{False}} \left( \begin{array}{c} \eta_0 \\ \eta_h \end{array} \right)}{} \end{array} \quad (30)$$

$$\begin{array}{c} \text{New-Epoch} \frac{\eta_e := \text{extraEntropy } pp}{pp \quad \eta_c \quad \eta_{ph} \vdash \left( \begin{array}{c} \eta_0 \\ \eta_h \end{array} \right) \xrightarrow[\text{TICKN}]{\text{True}} \left( \begin{array}{c} \eta_c \star \eta_h \star \eta_e \\ \eta_{ph} \end{array} \right)}{} \end{array} \quad (31)$$

**Figure 58:** Tick Nonce rules

## 12.5 Update Nonce Transition

The Update Nonce Transition updates the nonces until the randomness gets fixed. The environment is shown in Figure 59 and consists of the block nonce  $\eta$ . The update nonce state is shown in Figure 59 and consists of the candidate nonce  $\eta_c$  and the evolving nonce  $\eta_v$ .

The transition rule UPDN takes the slot  $s$  as signal. There are two different cases for UPDN: one where  $s$  is not yet StabilityWindow slots from the beginning of the next epoch and one where  $s$  is less than StabilityWindow slots until the start of the next epoch.

Note that in 32, the nonce candidate  $\eta_c$  transitions to  $\eta_v \star \eta$ , not  $\eta_c \star \eta$ . The reason for this is that even though the nonce candidate is frozen sometime during the epoch, we want the two nonces to again be equal at the start of a new epoch (so that the entropy added near the end of the epoch is not discarded).

*Update Nonce environments*

$$\text{UpdateNonceEnv} = ( \eta \in \text{Seed} \quad \text{new nonce} )$$

*Update Nonce states*

$$\text{UpdateNonceState} = \left( \begin{array}{ll} \eta_v \in \text{Seed} & \text{evolving nonce} \\ \eta_c \in \text{Seed} & \text{candidate nonce} \end{array} \right)$$

*Update Nonce Transitions*

$$\_ \vdash \_ \xrightarrow[\text{UPDN}]{\_} \_ \subseteq \mathbb{P} (\text{UpdateNonceEnv} \times \text{UpdateNonceState} \times \text{Slot} \times \text{UpdateNonceState})$$

**Figure 59:** UpdNonce transition-system types

$$\text{Update-Both} \frac{s < \text{firstSlot} ((\text{epoch } s) + 1) - \text{StabilityWindow}}{\eta \vdash \left( \begin{array}{l} \eta_v \\ \eta_c \end{array} \right) \xrightarrow[\text{UPDN}]{s} \left( \begin{array}{l} \eta_v \star \eta \\ \eta_v \star \eta \end{array} \right)} \quad (32)$$

$$\text{Only-Evolve} \frac{s \geq \text{firstSlot} ((\text{epoch } s) + 1) - \text{StabilityWindow}}{\eta \vdash \left( \begin{array}{l} \eta_v \\ \eta_c \end{array} \right) \xrightarrow[\text{UPDN}]{s} \left( \begin{array}{l} \eta_v \star \eta \\ \eta_c \end{array} \right)} \quad (33)$$

**Figure 60:** Update Nonce rules

## 12.6 Reward Update Transition

The Reward Update Transition calculates a new RewardUpdate to apply in a NEWEPOCH transition. The environment is shown in Figure 61, it consists of the produced blocks mapping  $b$  and the epoch state  $es$ . Its state is an optional reward update.

The transition rules are shown in Figure 62. There are three cases, one which computes a new reward update, one which leaves the rewards update unchanged as it has not yet been applied and finally one that leaves the reward update unchanged as the transition was started too early.

The signal of the transition rule RUPD is the slot  $s$ . The execution of the transition role is as follows:

*Reward Update environments*

$$\text{RUdpEnv} = \left( \begin{array}{ll} b \in \text{BlocksMade} & \text{blocks made} \\ es \in \text{EpochState} & \text{epoch state} \end{array} \right)$$

*Reward Update Transitions*

$$\_ \vdash \_ \xrightarrow[\text{RUPD}]{\_} \_ \subseteq \mathbb{P} (\text{RUdpEnv} \times \text{RewardUpdate}^? \times \text{Slot} \times \text{RewardUpdate}^?)$$

**Figure 61:** Reward Update transition-system types