SingularityNet Bonded Staking Pool Technical Specification

April 8, 2022

Contents

1 Introduction 1

2 Datums 1

3 Redeemers 2

4 Initialisation 2
- 4.1 Minting NFT State 2
- 4.2 Validator Parameters 2
- 4.3 Initiate Staking Pool 3

5 Bonded Staking Schema 3
- 5.1 User Stake 4
- 5.2 User Withdraw 4
- 5.3 Admin Close 5

1 Introduction

`Plutarch` is an eDSL in Haskell for writing on-chain scripts on Cardano. The intention is to write `SingularityNet`'s bonded stake pool code in `Plutarch` for optimised script size & execution units, thus reducing transaction fees compared to `PlutusTx`.

Production off chain code will be written in `Cardano Transaction Library (CTL)`, an API to balance and submit transactions using browser-integrated wallets and `Ogmios`.

Definition 1.1 (`NFT State Token`)**.** A common design pattern in Plutus contract involves minting an NFT with a (unique) currency symbol that parametrises the validator in question. This is just a utxo with the unit value for this currency symbol (and a fixed token name), the datum of which controls the state of the contract, we'll often refer to this UTXO as the **NFT State Token**.

Aim: to write a bonded staking pool for `SingularityNet`'s `AGIX` token.

2 Datums

```
data BondedStakingDatum
  = StakingStateDatum BondedStakingState
  | AssetDatum
```

Definition 2.1 (`StakingStateDatum`)**.** simply wraps `BondedStakingState` (in Definition 4.1) to become the datum of the state NFT token.

Definition 2.2 (`AssetDatum`)**.** is the datum for staked asset UTXOs at the script address.

3 Redeemers

```
data BondedStakingAction
   = StakeAct Natural PaymentPubKeyHash
   | WithdrawAct PaymentPubKeyHash
   | CloseAct
```

Definition 3.1 (`StakeAct`)**.** redeemer for staking tokens with `PaymentPubKeyHash` of `Natural` amount.

Definition 3.2 (`WithdrawAct`)**.** redeemer for withdrawing *all* staked tokens and rewards for a given `PaymentPubKeyHash`.

Definition 3.3 (`CloseAct`)**.** redeemer for the admin to close the stake pool after the withdrawal period has ended, withdrawing all possible leftover tokens.

4 Initialisation

4.1 Minting NFT State

Off-chain logic is required by the administrator/operator to initially mint an NFT with the following (unwrapped) datum (see 2.1 for the true datum):

```
data BondedStakingState = BondedStakingState
  { bss'stakees :: Map PaymentPubKeyHash Natural
  }
```

defining the state of validator/stake pool. The NFT token name can be hardcoded to "`BondedStaking`" or anything else, provided it's fixed for the codebase.

Definition 4.1 (`bss'stakees`)**.** A `Map` of stakees (`PaymentPubKeyHash`) to amount of tokens (`Natural`) to staked (excluding rewards) We will often refer to this as the **NFT Map**.

On-chain `Maps` can theoretically increase the transaction size to no end. `Plutarch` will drastically decrease script size allowing for a larger datum. We could heuristically provide a upper bound on the number of stakees per stake pool through local transaction size testing and incorporate into the validator parameters. Alternatively, we can attempt to implement an on-chain associated list. Would need to investigate how practical this is.

This spec will go with the former for now.

Ideally, we would have another token (minting policy) parameterised by: the validator address, the currency symbol of the original NFT and stakee's `PaymentPubKeyHash`. The datum of which would carry information on staked amount for that user. This would be verified in the user staking/deposit step (see Subsection 5.1). However, I don't believe the validator can call a minting policy onchain to verify the currency symbol is of the right user (atleast in `PlutusTx`). Investigate whether possible with `Plutarch`, seems unlikely?

4.2 Validator Parameters

The currency symbol of the NFT then parametrises the validator as follows:

```
data BondedPoolParams = BondedPoolParams
  { bpp'duration :: POSIXTimeRange
  , bpp'interest :: NatRatio
  , bpp'size :: Natural
  , bpp'rewards :: Natural
  , bpp'contributionPeriod :: POSIXTimeRange
  , bpp'withdrawalPeriod :: POSIXTimeRange
  , bpp'minStake :: Natural
  , bpp'maxStake :: Natural
  , bpp'operator :: PaymentPubKeyHash
  , bpp'bondedAssetClass :: AssetClass
  , bpp'bondedStakingStateCs :: CurrencySymbol -- this uniquely parameterises the
  -- validator
  , bpp'maxStakees :: Natural
 }
```

The parameters are configurable by the administrator/operator at the start and fixed for the duration of the staking period.

Definition 4.2 (`bpp'bondedAssetClass`)**.** the asset class of the token being staked, i.e. `AGIX`. Perhaps `CurrencySymbol` will suffice.

Definition 4.3 (`bpp'duration`)**.** total duration of staking pool.

Definition 4.4 (`bpp'interest`)**.** a positive (non-zero) ratio fixed decimal. This is the resulting `APY` obtainable offchain for a fixed period rate, r and number of compound periods, n.

Definition 4.5 (`bpp'size`)**.** maximum number of tokens allowed in the staking pool.

Definition 4.6 (`bpp'rewards`)**.** maximum pre-determined rewards, calculated as (after safely coercing) *ceiling*(`bpp'interest` $*$ `bpp'size`) rounding up to ensure we have enough tokens. This is part of the parameters for on-chain convenience.

Definition 4.7 (`bpp'contributionPeriod`)**.** duration in which wallets can contribute to the pool.

Definition 4.8 (`bpp'withdrawalPeriod`)**.** duration in which wallets can withdraw from the pool.

Definition 4.9 (`bpp'minStake`)**.** minimum amount required to stake by a wallet.

Definition 4.10 (`bpp'maxStake`)**.** maximum amount possible to stake by a wallet.

Definition 4.11 (`bpp'operator`)**.** the `PaymentPubKeyHash` of the administrator.

Definition 4.12 (`bpp'bondedStakingStateCs`)**.** currency symbol of the NFT to identify the state of the pool, see `BondedStakingState`.

Definition 4.13 (`bpp'maxStakees`)**.** maximum number of stakees for a staking pool.

It would be sensible to make sure the contribution and withdrawal periods do not overlap, with the latter *after* the former.

4.3 Initiate Staking Pool

The following steps should be taken to initiate the staking pool

- The minted NFT from Subsection 4.1 should be sent to the validator address determined by Subsection 4.2
- The maximum reward tokens `bpp'rewards` from Definition 4.6 should be sent to the same validator address

5 Bonded Staking Schema

In this set up, the redeemer acts are in one-to-one correspondence with the off-chain schema. Extra functionality like querying the stake pool is possible too.

5.1 User Stake

On-chain conditions for spending `BondedStakingState` UTXO with `StakeAct` redeemer (see Definition 3.1), if any of these conditions do not hold, validation should fail:

- Signed by `PaymentPubKeyHash` from the redeemer
- Transaction must occur within the contribution period, see Definition 4.7
- Check the requested stake amount is positive (since `Natural` includes zero)
- Check the requested stake amount plus any amount the user has already staked (according to `bss'stakees`, the NFT `Map`) is between the minimum and maximum allowed amount (inclusive of bounds say), see Definitions 4.9, 4.10. This enables users to stake multiple times during the contribution period
- Check the NFT `Map` from Definition 4.1 has been incremented accordingly. A user can only increment their amount and no one elses
- Check the pool size (Definition 4.5) will not exceeded by the redeemer staking amount. We can compare with the state NFT `Map`
- Check max stakees (Definition 4.13) will not be exceeded by the current redeemer
- Check the NFT is part of the inputs and (continuing) outputs, sent back to the validator address, where the datum is incremented exactly as described above. We can find the NFT by using `bpp'bondedStakingStateCs` (see Definition 4.12) and our hardcoded token name
- Check the correct amount of the bonded asset class (see Definition 4.2) is deposited to the validator address with `AssetDatum` as datum (see Definition 2.2)

The above conditions should be mirrored in off chain code so that validation will pass.

5.2 User Withdraw

On-chain conditions for spending `BondedStakingState` and asset UTXOs with `WithdrawAct` redeemer (see Definition 3.2), if any of these conditions do not hold, validation should fail:

- Signed by `PaymentPubKeyHash` from the redeemer
- Transaction must occur within the withdrawal period, see Definition 4.8
- Check the NFT `Map` from Definition 4.1 has removed the user from the `Map`. A user can only remove themselves and no one else. Ensure the `Map` is otherwise unchanged
- Check the NFT is part of the inputs and (continuing) outputs, sent back to the validator address, where datum changes are described exactly as above. We can find the NFT by using `bpp'bondedStakingStateCs` (see Definition 4.12) and our hardcoded token name.
- Check the correct amount (given by the NFT `Map` before removing) plus rewards, of the bonded asset class (see Definition 4.2) is withdrawn to the user address

The total withdrawal amount for a given user is $s + s * interest$, where s is the staked amount from NFT `Map`. We need to be careful with rounding behaviour when withdrawing as the requested withdrawal amount off chain must match what the validator expects.

The above conditions should be mirrored in off chain code so that validation will pass.

5.3 Admin Close

On-chain conditions for spending `BondedStakingState` UTXO with `CloseAct` redeemer (see Definition 3.2), if any of these conditions do not hold, validation should fail. We should split the logic into two paths:

1. The withdrawal period has a **finite** upperbound, in which case the administrator can withdraw all leftover tokens including the NFT state token after the withdrawal period has ended:

    - Signed by operator from validator parameters, see Definition 4.11
    - Transaction must occur after the end of the total duration and withdrawal period, see Definitions 4.3, 4.8

    On top of the mentioned validator conditions, the offchain code only requires that leftover UTXOs are spent as balancing automatically sends them back to the administrator wallet.

2. The withdrawal period is **positive infinity**, in which case the administrator can only withdraw unclaimed rewards. The NFT state token can only be withdrawn after all claimed rewards are withdrawn, since it is required by stakees to withdraw:

    - Signed by operator from validator parameters, see Definition 4.11
    - Transaction must occur after the end of the total duration, see Definition 4.3
    - If the NFT `Map` is empty (so all withdrawals have occurred), no further checks are required, the administrator can withdraw everything. If the NFT `Map` still contains outstanding rewards, the administrator can withdraw: $\texttt{bpp'rewards} - \texttt{bpp'interest} * \sum_i \texttt{NftMapValue}_i$ (with suitable rounding) of the underlying staked token and **not** the NFT state token with `bpp'bondedStakingStateCs` currency symbol

    The above conditions should be mirrored in off chain code so that validation will pass.