Formal Verification of Daoism ProtocolFeeSplitter (September-October 2022)

Summary

This document describes the specification and verification of Daoism's ProtocolFeeSplitter contract on top of Balancer pools using the Certora Prover. The work was undertaken from Sept 27th to Oct 11th. The latest commit that was reviewed and run through the Certora Prover was commit f5e9c78.

The scope of our verification was the following contracts and their various components:

- [ProtocolFeeSplitter.sol]

The Certora Prover proved the implementation of the ProtocolFeeSplitter contract is correct with respect to the formal rules written by the Daoism and the Certora teams. During the verification process, the Certora Prover did not discover bugs.

Disclaimer

The Certora Prover takes as input a contract and a specification and formally proves that the contract satisfies the specification in all scenarios. Importantly, the guarantees of the Certora Prover are scoped to the provided specification, and the Certora Prover does not check any cases not covered by the specification.

We hope that this information is useful, but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.
**Notations**

- ✔️ indicates the rule is formally verified on the latest reviewed commit. We write ✔️* when the rule was verified on a simplified version of the code (or under some assumptions).

- ❌ indicates the rule was violated under one of the tested versions of the code.

- 🔥 indicates the rule has not been checked on the current version.

- ☹️ indicates that some functions cannot be verified because the rules timed out.

Footnotes describe any simplifications or assumptions used while verifying the rules (beyond the general assumptions listed above).

**Verification of ProtocolFeeSplitter.sol**

**Assumptions**

The `ProtocolFeeSplitter` contract is built on top of Balancer pools. It is a simple contract consisting of three setters, one calculation function and some auxiliary permission functions that works with Balancer's authentication mechanism.

For rules regarding revenue sharing fee percentage, we make the following assumptions about the state variables of the system:

- `defaultRevenueSharingFeePercentage` is between 0 and `_MAX_REVENUE_SHARING_FEE_PERCENTAGE` (set in the contract as 50%)
- `revenueSharePercentageOverride` is between `_MIN_REVENUE_SHARING_FEE_PERCENTAGE` (set in the contract as 1%) and `_MAX_REVENUE_SHARING_FEE_PERCENTAGE` (set in the contract as 50%)
- The `getOwner()` function has been set to return a specific value for a given poolId.

**Harnessing**

In order to give the Prover access to internal state, we verified an extension of the contract (`ProtocolFeeSplitterHarness.sol`) that added new methods. We harnessed `_canPerform(bytes32, address)` to return a constant boolean for given inputs, thus bypassing the need of an external contract in Balancer's authentication mechanism.

**Properties**
(✓) invariant RevenueSharingFeePercentageInRange:
RevenueSharingFeePercentageInRange(bytes32 poolId) checks
revenueSharePercentageOverride of the given poolId is always between
_MIN_REVENUE_SHARING_FEE_PERCENTAGE and _MAX_REVENUE_SHARING_FEE_PERCENTAGE.

(✓) invariant DefaultRevenueSharingFeePercentageInRange:
DefaultRevenueSharingFeePercentageInRange() checks
DefaultRevenueSharingFeePercentageInRange is always between 0 and
_MAX_REVENUE_SHARING_FEE_PERCENTAGE.

(✓) rule SetRevenueSharingFeePercentageCorrectly:
setRevenueSharingFeePercentage(bytes32 poolId, uint256 newSwapFeePercentage)
reverts if the caller is not authenticated or newSwapFeePercentage is out-of-range,
otherwise it set revenueSharePercentageOverride for the given poolId.

(✓) rule SetDefaultRevenueSharingFeePercentageCorrectly:
setDefaultRevenueSharingFeePercentage(uint256 feePercentage) reverts if the caller
is not authenticated or input is out-of-range. Otherwise it sets
defaultRevenueSharingFeePercentage.

(✓) rule SetPoolBeneficiaryCorrectly:
setPoolBeneficiary(bytes32 poolId, address newBeneficiary) reverts if the msg.sender is not the pool owner for the
specified poolId, otherwise it sets beneficiary for the poolId.

(✓) rule CollectFeesRevertCondition:
collectFees(bytes32 poolId) reverts when there are no fees to be collected.

(✓) rule CollectFeesIntegrity:
collectFees(bytes32 poolId) splits
feeCollectorBptBalance between the beneficiary of the pool and the treasury. If
beneficiary is not set, all fees go to treasury. Collecting fees does not change the
total supply of the pool token.

(✓) rule ZeroFeeCollectorBptBalanceAfterCollectFee:
After collectFees(bytes32 poolId) is called successfully, there are be no fees balance remaining.

(✓) rule FeeCollectorBptBalanceShouldNotChange:
No functions other than
collectFees(bytes32 poolId) change the fees balance.

The above properties can be found here