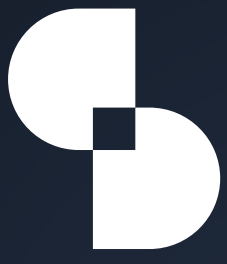


Formal Verification of Silo V1, July 2022

Table of Contents

1. Summary

This document describes the specification and verification of Silo's protocol using the Certora Prover. The work was performed between May 7, 2022 to Jul. 15, 2022 while the code was still in development.

The scope of this verification is Silo's protocol and contracts related to it:

- /lib/EasyMath.sol
- /lib/Solvency.sol
- /priceProviders/balancerV2/BalancerV2PriceProvider.sol
- /priceProviders/uniswapV3/UniswapV3PriceProvider.sol
- /utils/GuardedLaunch.sol
- /utils/Manageable.sol
- /utils/ShareCollateralToken.sol
- /utils/ShareDebtToken.sol
- /utils/TwoStepOwnable.sol
- InterestRateModel.sol
- PriceProvidersRepository.sol
- Silo.sol
- SiloFactory.sol
- SiloRepository.sol
- TokensFactory.sol

The Certora Prover proved the implementation of the protocol is correct with respect to formal specifications written by the Silo team and reviewed by the Certora team.

2. List of main issues discovered

Severity: High

Issue:	**Accrued interest lost while withdrawing assets**
Description:	In the withdraw function total deposits been rewritten by liquidity value which doesn't contain accrued interest..
Properties violated:	Silo valid states properties.
Mitigation/Fix:	Update total deposits properly in the withdraw function.

Severity: High

Issue:	**Accrue interest overflow if compounded interest achieves RCOMP_MAX.**
Description:	Interest rate model was secured to handle overflow cases. In a Silo with critical utilisation ratio, interest rate can increase significantly generating large compounded interest (max growth of interest rate is proportional to the square of time difference in seconds). RCOMP_MAX was set right before the overflow of the exp(x) function. In BaseSilo _accrueInterest modifier multiplies totalBorrowAmount to the value of rcomp. High threshold for compounded interest caused these intermediC ate calculations to overflow with totalBorrowAmount close to 10^18. Revert of _accrueInterest makes it impossible to withdraw collateralOnly deposits or to liquidate insolvent borrow positions.
Properties violated:	Interest model unit tests.
Mitigation/Fix:	RCOMP_MAX is set to the lower value, the interest rate model is less aggressive on long term stale periods. BaseSilo is fixed to handle these overflow cases (the probability of those scenarios is insignificant).

2. List of main issues discovered *(cont.)*

Severity: Medium

Issue:	**Withdrawal for free because of a rounding issue**
Description:	Zero shares burned but some amount was withdrawn.
Properties violated:	Silo high level properties.
Mitigation/Fix:	Revert in the EasyMath if the amount is not 0 but the result is.

Severity: Medium

Issue:	**Rounding in a favor of the protocol**
Description:	Redeeming deposited tokens didn't burn equally proportional share tokens due to a rounding issue in solidity.
Properties violated:	Silo high level properties.
Mitigation/Fix:	Round in a favor of the protocol.

Severity: Medium

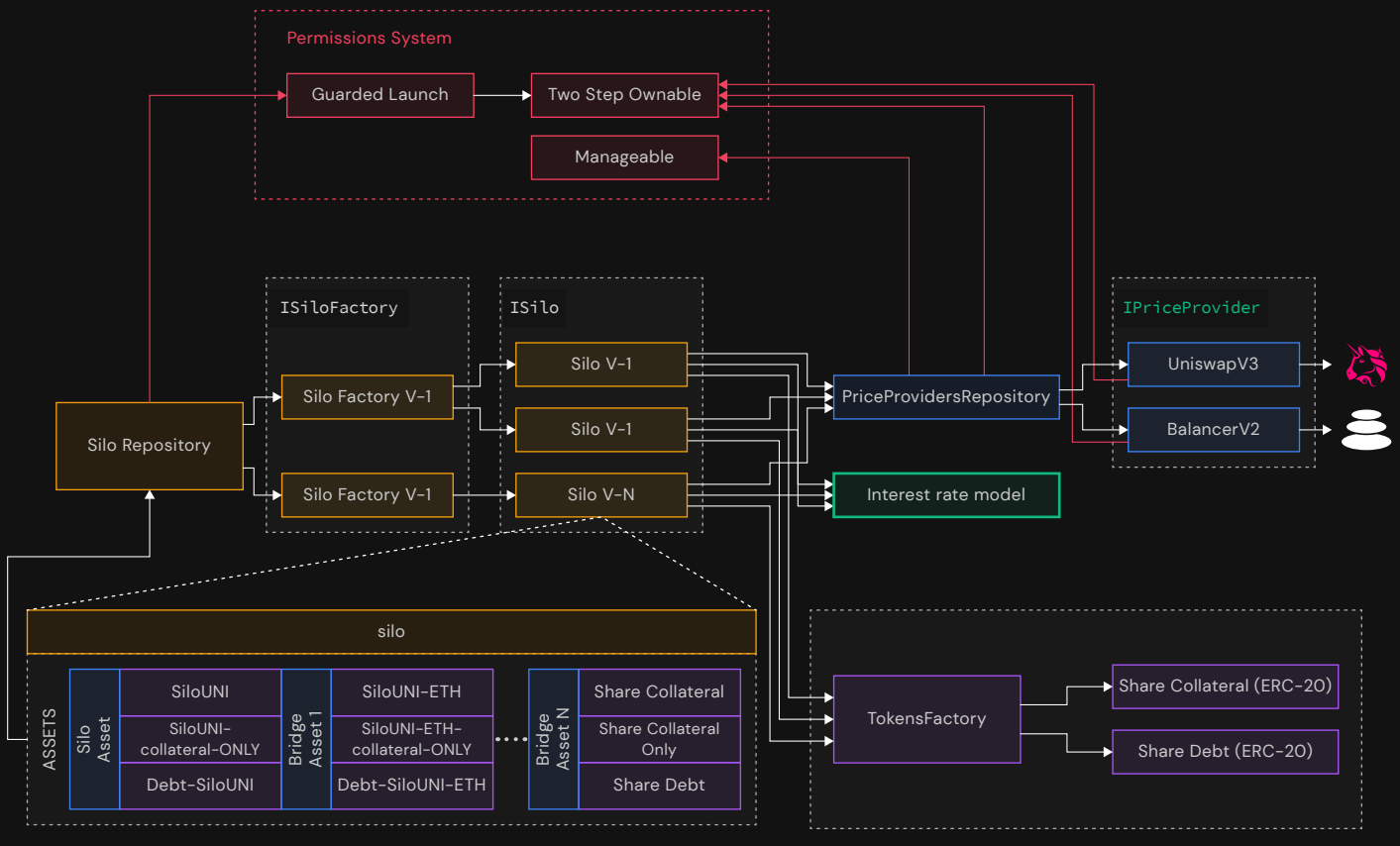
Issue:	**Missed validation for the interest rate model config**
Description:	Because of the lack of the config validation from the smart contracts side, there was a possibility to turn the interest model into an extreme state.
Properties violated:	Interest model variable changes properties.
Mitigation/Fix:	Added config validation.

3. Summary of Formal Verification

Overview of Silo Protocol

Silo is an isolated-market lending protocol. Smart contracts have a modular design and are mostly following Uniswap's naming convention. The protocol consists of multiple components, shown on *Fig. 1 Silo protocol architecture*.

Fig. 1 Silo protocol architecture



PriceProvidersRepository

The role of an oracle is to provide Silo with the correct price of an asset. SiloOracleRepository is the entry point of token prices for a Silo and manages oracle modules and price request routing. It can support many protocols and sources.

3. Summary of Formal Verification *(cont.)*

BalancerV2PriceProvider

BalancerV2Oracle is an oracle module that is responsible for pulling the correct prices of a given asset from BalancerV2 pools. It performs security checks and returns TWAP prices when requested.

UniswapV3PriceProvider

UniswapV3Oracle is an oracle module that is responsible for pulling the correct prices of a given asset from UniswapV3 pools. It performs security checks and returns TWAP prices when requested.

Silo

Silo is the main component of the protocol. It implements lending logic, manages and isolates risk, acts as a vault for assets, and performs liquidations. Each Silo is composed of the base asset for which it was created (e.g. UNI) and bridge assets (e.g. ETH and SiloDollar). There may be multiple bridge assets at any given time.

SiloRepository

Repository handles the creation and configuration of Silos.

- Stores configuration for each asset in each Silo: Each asset in each Silo starts with a default config that later on can be changed by the contract owner.
- Stores registry of Factory contracts that deploy different versions of Silos: It is possible to have multiple versions/implementations of Silo and use different versions for different tokens. For example, one version can be used for UNI (ERC20) and the other can be used for UniV3LP tokens (ERC721).
- Manages bridge assets: Each Silo can have 1 or more bridge assets. New Silos are created with all currently active bridge assets. Silos that are already developed must synchronize bridge assets. Sync can be done by anyone since the function has public access.
- Is a single source of truth for other contract addresses.

3. Summary of Formal Verification *(cont.)*

SiloFactory

Factory contract performs deployment of each Silo. Many Factory contracts can be registered with the Repository contract.

Interest Rate Model

The Interest Rate Model calculates the dynamic interest rate for each asset (base asset and bridge assets) in each Silo at any given time. The model calculates two values:

- Current Interest Rate: Used to display the current interest rate for the user in UI.
- Compound Interest Rate: Returns the interest rate for a given time range compounded every second.

4. Assumption and simplification made during verification

We made the following assumptions during the verification process:

- Assume a 1:2 ratio share per amount for the Silo properties.
- Assume that the asset price is always 4.
- Implemented a Silo function selector where functions that can perform an action with interest calculation and without.
- Implemented a simplified tokens factory for Silo tests.
- When verifying contracts that make external calls, we assume that those calls can have arbitrary side effects outside of the contracts but that they do not affect the state of the contract being verified. This means that some reentrancy bugs may not be caught.
- Implemented 'harness' contracts to be able to test libraries and abstract contracts or add additional getters that are required for rules implementation.
- Overflow cases in compounded interest and accrued interest intermediate calculations are skipped in interest rate model unit tests and high level mathematical properties. Overflow cases are handled to prevent transaction reverts; overflowable values will be set to its top limits. These limitations break continuous mathematical properties in the interest rate model long term. These properties are verified on core implementation with skipped overflow edge cases. All interest rate model properties hold in short term interest compounding periods (interest rate model compounded interest update time less than 19 days for total borrowed amount less or equal 10^25 wei).

5. 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 that are not covered by the specification.

The purpose of this report is informational only and should not be construed as explicit or implied guarantee of the security of Silo's smart contracts and codebase.

6. Verifications

6.1 Easy Math

Reports: EasyMath

6.1.1 Math Properties

6.1.1.1 Amount to shares conversion is monotonic.
Implementation: rule MP_monotonicity_amount_toShares
6.1.1.2 Shares to amount conversion is monotonic.
Implementation: rule MP_monotonicity_shares_toAmount
6.1.1.3 Inverse conversion for amount returns value less or equal to the amount.
Implementation: rule MP_inverse_amount
6.1.1.4 Inverse conversion for shares returns value less or equal to the shares.
Implementation: rule MP_inverse_shares

6.1.2 Risk Assessment

6.1.2.1 If the deposit was made when total deposits were equal to the total shares, after gaining any interest, there should not be scenarios where the withdrawal amount will be less than the deposited amount.
Implementation: rule RA_withdraw_with_interest

6.2 Interest Rate Model

6.2.1 Valid States

Report: Valid States

6.2.1.1 Decimal points are 10^18 and can not be changed.
Implementation: rule VS_DP
6.2.1.2 RCOMP_MAX is equal to (2^16) * 10^18 and can not be changed.
Implementation: rule VS_RCOMP_MAX
6.2.1.3 X_MAX is equal to 110903701476317733 13 (X_MAX ≈ ln(RCOMP_MAX + 1)) and can not be changed
Implementation: rule VS_X_MAX
6.2.1.4 For every Silo and every asset Config.uopt ∈ (0, 10^18) in DP.
Implementation: rule VS_uopt
6.2.1.5 For every Silo and every asset Config.ucrit ∈ (uopt, 10^18) in DP.
Implementation: rule VS_ucrit

6.2.1.6 For every Silo and every asset Config.ulow ∈ (0, uopt) in DP.
Implementation: rule VS_ulow
6.2.1.7 For every Silo and every asset Config.ki > 0 (integrator gain).
Implementation: rule VS_ki
6.2.1.8 For every Silo and every asset Config.kcrit > 0 (proportional gain for large utilization).
Implementation: rule VS_kcrit
6.2.1.9 For every Silo and every asset Config.klow ≥ 0 (proportional gain for low utilization).
Implementation: rule VS_klow
6.2.1.10 For every Silo and every asset Config.klin ≥ 0 (coefficient of the lower linear bound).
Implementation: rule VS_klin
6.2.1.11 For every Silo and every asset Config.beta ≥ 0.
Implementation: rule VS_beta
6.2.1.12 For every Silo and every asset Config.ri ≥ 0.
Implementation: rule VS_complexInvariant_ri
6.2.1.13 For every Silo and every asset Config.tcrit ≥ 0.
Implementation: rule VS_complexInvariant_tcrit
6.2.1.14 ASSET_DATA_OVERFLOW_LIMIT is equal to (2^196) and can not be changed.
Implementation: rule VS_ASSET_DATA_OVERFLOW_LIMIT

6.2.2 Variable Changes
Report: Variable Changes

6.2.2.1 Config.uopt can be set only by setConfig. ∀ Silo ∀ Asset ((uopt changed) <=> (f.selector == setConfig && msg.sender == owner)).
Implementation: rule VCH_uoptChangedOnlyOwner
6.2.2.2 Config.ucrit can be set only by setConfig. ∀ Silo ∀ Asset ((ucrit changed) <=> (f.selector == setConfig && msg.sender == owner)).
Implementation: rule VCH_ucritChangedOnlyOwner
6.2.2.3 Config.ulow can be set only by setConfig. ∀ Silo ∀ Asset ((ulow changed) <=> (f.selector == setConfig && msg.sender == owner)).
Implementation: rule VCH_ulowChangedOnlyOwner
6.2.2.4 Config.ki can be set only by setConfig. ∀ Silo ∀ Asset ((ki changed) <=> (f.selector == setConfig && msg.sender == owner)).
Implementation: rule VCH_kiChangedOnlyOwner

6.2.2.5 Config.kcrit can be set only by setConfig. ∀ Silo ∀ Asset ((kcrit changed) <=> (f.selector == setConfig && msg.sender == owner)).
Implementation: rule VCH_kcritChangedOnlyOwner
6.2.2.6 Config.klow can be set only by setConfig. ∀ Silo ∀ Asset ((klow changed) <=> (f.selector == setConfig && msg.sender == owner)).
Implementation: rule VCH_klowChangedOnlyOwner
6.2.2.7 Config.klin can be set only by setConfig. ∀ Silo ∀ Asset ((klin changed) <=> (f.selector == setConfig && msg.sender == owner)).
Implementation: rule VCH_klinChangedOnlyOwner
6.2.2.8 Config.beta can be set only by setConfig. ∀ Silo ∀ Asset ((beta changed) <=> (f.selector == setConfig && msg.sender == owner)).
Implementation: rule VCH_betaChangedOnlyOwner
6.2.2.9 Config.ri can be set only by setConfig or by getCompoundInterestRateAndUpdate. ∀ Silo ∀ Asset ((ri changed) <=> (f.selector == setConfig && msg.sender == owner || f.selector == getCompoundInterestRateAndUpdate && msg.sender == silo)).
Implementation: rule VCH_riChangedOnlyOwnerOrInterestUpdate
6.2.2.10 Config.tcrit can be set only by setConfig or by getCompoundInterestRateAndUpdate. ∀ Silo ∀ Asset ((tcrit changed) <=> (f.selector == setConfig && msg.sender == owner || f.selector == getCompoundInterestRateAndUpdate && msg.sender == silo)).
Implementation: rule VCH_tcritChangedOnlyOwnerOrInterestUpdate

6.2.3 Unit Tests
Reports: Compound Interest Rate, Current Interest Rate

6.2.3.2 CalculateCompoundInterestRate. tcrit and ri were in a state before the function call. Utilisation before the call was u. tcritNew, riNew and rcomp are the return values.

- Assert (u > Config.ucrit && Config.beta != 0) <=> (tcritNew > tcrit).
- Assert (u > Config.uopt) => (riNew >= ri).
- Assert (u > Config.uopt) && (ri <= Config.klin * u / DP()) => (riNew >= Config.klin * u / DP()).
- Assert (u == Config.uopt) && (ri < Config.klin * u / DP()) => (riNew == Config.klin * u / DP()).
- Assert (u == Config.uopt) && (ri >= Config.klin * u / DP()) => (riNew == ri).

- Assert (u <= Config.uopt) && (ri <= Config.klin * u / DP()) => (riNew == Config.klin * u / DP()).
- Assert (u < Config.uopt) && (ri > Config.klin * u / DP()) => (riNew <= ri) && (riNew >= Config.klin * u / DP()).

*Implementation: rule UT_calculateCompoundInterestRate_**

6.2.3.3 GetCurrentInterestRate. For two consecutive block timestamps tNew > tOld. Let uOld is utilisation ratio at tOld timestamp, rCurOld is current interest rate at tOld. Let uNew is utilisation ratio at tNew timestamp, rCurNew is current interest rate at tNew.

- Assert (uOld < uNew) && (rCurOld <= Config.klin * uOld / DP()) => (rCurNew >= rCurOld).
- Assert (uOld > Config.uopt && uNew > uOld) => (rCurNew >= rCurOld).
- Assert (uOld >= uNew) && (rCurNew > rCurOld) => (uOld >= Config.uopt).
- Assert (rCurNew == 0) => (u * Config.klin / DP() == 0).

*Implementation: rule UT_calculateCurrentInterestRate_**

6.2.3.4 Max. a >= b <=> max(a, b) returns a.

Implementation: rule UT_max

6.2.3.5 Min. a <= b <=> min(a, b) returns a.

Implementation: rule UT_min

6.2.4 High Level Properties

These properties were proven by the Certora team using the fuzzy mining feature for solving complex problems.

6.2.4.1 rComp is the current output of getCompoundInterestRate, rCurNew is the current interest rate, uNew is the current utilisation ratio, T is the difference between the last interest rate update timestamp and current timestamp. Assert (u <= Config.uopt) => (rComp >= rCurNew * T).

Implementation: rule PMTH_compoundAndCurrentInterest_uGreaterUopt

6.2.4.2 rComp is the current output of getCompoundInterestRate, rCurOld is the interest rate on the last interest rate update timestamp, uNew is the current utilisation ratio, T is the difference of the last interest rate update timestamp and current timestamp. Assert (u >= Config.uopt) => (rComp >= rCurOld * T).

Implementation: rule PMTH_compoundAndCurrentInterest_uLessUopt

6. Verifications *(cont.)*

6.3 Permissions

6.3.1 Manageable
Report: Manageable

6.3.1.1 Only changeManager can set a manager.
Implementation: rule VC_manager_change
6.3.1.2 A manager can't be an empty address.
Implementation: rule VS_manager_is_not_0
6.3.1.3 Only the owner or the manager can execute changeManager.
Implementation: rule VS_changeManager_only_owner_or_manager

6.3.2 Two Steps Ownable
Reports: TwoStepOwnable

6.3.2.1 Only renounceOwnership can set an owner.
Implementation: rule VC_owner_to_0
6.3.2.2 Only transferOwnership, renounceOwnership and acceptOwnership can update an owner.
Implementation: rule VC_owner_update
6.3.2.3 Only acceptOwnership, renounceOwnership, transferOwnership, removePendingOwnership
can set a pending owner to an empty address.
Implementation: rule VC_pending_owner_to_0
6.3.2.4 Only transferPendingOwnership can set a pending owner.
Implementation: rule VC_pending_owner_config
6.3.2.5 If an owner is an empty address, a pending owner should also be an empty address.
Implementation: rule VS_empty_state
6.3.2.6 If the owner is updated, a pending owner should be an empty address.
Implementation: rule VS_owner_update
6.3.2.7 Only the owner can execute renounceOwnership.
Implementation: rule VS_renounceOwnership_only_owner
6.3.2.8 Only the owner can execute transferOwnership.
Implementation: rule VS_transferOwnership_only_owner
6.3.2.9 Only the owner can execute transferPendingOwnership.
Implementation: rule VS_transferPendingOwnership_only_owner

6.3.2.10 Only the owner can execute removePendingOwnership.
Implementation: rule VS_removePendingOwnership_only_owner
6.3.2.11 Only the pending owner can execute acceptOwnership.
Implementation: rule VS_acceptOwnership_only_pending_owner

6.4 Price Providers

6.4.1 BalancerV2
Reports: BalancerV2PriceProvider

6.4.1.1 An asset pool can be configured only by setupAsset fn.
Implementation: rule VC_BalancerV2_asset_pool
6.4.1.2 _state.periodForAvgPrice can be updated only by changePeriodForAvgPrice, changeSettings.
Implementation: rule VC_BalancerV2_periodForAvgPrice
6.4.1.3 _state.secondsAgo can be updated only by changeSecondsAgo, changeSettings.
Implementation: rule VC_BalancerV2_secondsAgo
6.4.1.4 _state.periodForAvgPrice can't be set to 0
Implementation: rule VS_BalancerV2_periodForAvgPrice_is_not_zero
6.4.1.5 Only the manager can configure an asset pool.
Implementation: rule UT_BalancerV2_setupAsset_only_manager
6.4.1.6 Only the manager can configure a periodForAvgPrice.
Implementation: rule UT_BalancerV2_changePeriodForAvgPrice_only_manager
6.4.1.7 Only the manager can configure a secondsAgo.
Implementation: rule UT_BalancerV2_changeSecondsAgo_only_manager
6.4.1.8 Only the manager can change settings.
Implementation: rule UT_BalancerV2_changeSettings_only_manager
6.4.1.9 getPrice fn should revert if a Price oracle is not configured for an asset.
Implementation: rule UT_BalancerV2_getPrice_with_not_configured_pool

6.4.2 Price Providers Repository
Reports: PriceProvidersRepository

6.4.2.1 Add to _allProviders array can only addPriceProvider.
Implementation: rule VC_Price_providers_repository_add_provider
6.4.2.2 Remove from _allProviders array can only removePriceProvider.
Implementation: rule VC_Price_providers_repository_remove_provider

6.4.2.3 Change priceProviders can only setPriceProviderForAsset.
Implementation: rule VC_Price_providers_repository_priceProviders
6.4.2.4 Only the owner can add the price provider.
Implementation: rule UT_Price_providers_repository_add_provider
6.4.2.5 Only the owner can remove the price provider.
Implementation: rule UT_Price_providers_repository_remove_provider
6.4.2.6 Only the owner can set the price provider for an asset.
Implementation: rule UT_Price_providers_repository_set_provider

6.4.3 UniswapV3
Reports: UniswapV3 price provider

6.4.3.1 An asset pool can be configured only by setupAsset fn.
Implementation: rule VC_UniswapV3_asset_pool
6.4.3.2 priceCalculationData.periodForAvgPrice can be updated only by changePeriodForAvgPrice fn.
Implementation: rule VC_UniswapV3_periodForAvgPrice
6.4.3.3 priceCalculationData.blockTime can be updated only by changeBlockTime fn.
Implementation: rule VC_UniswapV3_blockTime
6.4.3.4 Only the manager can configure an asset pool.
Implementation: rule UT_UniswapV3_setupAsset_only_manager
6.4.3.5 Only the manager can configure a periodForAvgPrice.
Implementation: rule UT_UniswapV3_changePeriodForAvgPrice_only_manager
6.4.3.6 Only the manager can configure a blockTime.
Implementation: rule UT_UniswapV3_changeBlockTime_only_manager

6.5 Shares Tokens

6.5.1 Shares Tokens Common Properties
Reports: Shares tokens risk assessment, Shares tokens unit tests, Common shares tokens high level props, Common shares tokens variable changes

6.5.1.1 TotalSupply can only change on mint, burn.
Implementation: rule `VC_Shares_totalSupply_change`
6.5.1.2 TotalSupply can increase only on mint.
Implementation: rule `VC_Shares_totalSupply_increase`

6.5.1.3 TotalSupply can decrease only on burn.
Implementation: rule `VC_Shares_totalSupply_decrease`
6.5.1.4 For any address, the balance can change only on mint, burn, transfer, transferFrom.
Implementation: rule `VC_Shares_balance_change`
6.5.1.5 For any address, the balance can increase only on mint, transfer, transferFrom.
Implementation: rule `VC_Shares_balance_increase`
6.5.1.6 For any address, the balance can decrease only on burn, transfer, transferFrom.
Implementation: rule `VC_Shares_balance_decrease`
6.5.1.7 Allowance can only change on transferFrom, approve, increaseAllowance, decreaseAllowance.
Implementation: rule `VC_Shares_allowance_change`
6.5.1.8 Sum of all balances should be equal totalSupply.
Implementation: invariant `VS_Shares_totalSupply_balances`
6.5.1.9 transferFrom should decrease allowance for the same amount as transferred.
Implementation: rule `HLP_Shares_transferFrom_allowance`
6.5.1.10 Additive transfer. Balance change for msg.sender and recipient while do transfer($amount$) should be the same as transfer($amount/2$) + transfer($amount/2$).
Implementation: rule `HLP_Shares_additive_transfer`
6.5.1.11 Additive transferFrom. Balance change for sender and recipient while do transferFrom($amount$) should be the same as transferFrom($amount/2$) + transferFrom($amount/2$).
Implementation: rule `HLP_Shares_additive_transferFrom`
6.5.1.12 Additive mint. Balance change for recipient while do mint($amount$) should be the same as mint($amount/2$) + mint($amount/2$).
Implementation: rule `HLP_Shares_additive_mint`
6.5.1.13 Additive burn. Balance change for recipient while do burn($amount$) should be the same as burn($amount/2$) + burn($amount/2$).
Implementation: rule `HLP_Shares_additive_burn`
6.5.1.14 Additive increaseAllowance. Allowance change for spender while do increaseAllowance($amount$) should be the same as increaseAllowance($amount/2$) + increaseAllowance($amount/2$).
Implementation: rule `HLP_Shares_additive_increaseAllowance`

6.5.1.15 Additive decreaseAllowance. Allowance change for spender while do decreaseAllowance($amount$) should be the same as decreaseAllowance($amount/2$) + decreaseAllowance($amount/2$).
Implementation: rule `HLP_Shares_additive_decreaseAllowance`
6.5.1.16 Integrity of mint. Balance of recipient after mint($amount$) should be equal to the balance of the recipient before mint + $amount$.
Implementation: rule `HLP_Shares_integrity_mint`
6.5.1.17 Integrity of burn. Balance of recipient after burn($amount$) should be equal to the balance of the recipient before burn - $amount$.
Implementation: rule `HLP_Shares_integrity_burn`
6.5.1.18 Integrity of transfer. Balance of recipient and msg.sender after transfer($amount$) should be updated for the exact amount that has been requested for a transfer.
Implementation: rule `HLP_Shares_integrity_transfer`
6.5.1.19 Integrity of transferFrom. Balance of recipient and sender after transferFrom($amount$) should be updated for the exact amount that has been requested for a transferFrom.
Implementation: rule `HLP_Shares_integrity_transferFrom`
6.5.1.20 Integrity of increaseAllowance. Allowance of spender after increaseAllowance($amount$) should be equal to the allowance of the spender before increaseAllowance + $amount$.
Implementation: rule `HLP_Shares_integrity_increaseAllowance`
6.5.1.21 Integrity of decreaseAllowance. Allowance of spender after decreaseAllowance($amount$) should be equal to the allowance of the spender before decreaseAllowance - $amount$.
Implementation: rule `HLP_Shares_integrity_decreaseAllowance`
6.5.1.22 Integrity of approve. Allowance of spender after approve($amount$) should be equal to the allowance of the spender before approve + $amount$.
Implementation: rule `HLP_Shares_integrity_approve`
6.5.1.23 Mint and Burn should revert if the sender is not the silo address.
Implementation: rule `UT_Shares_min_burn_permissions`
6.5.1.24 Each action affects at most two users' balance.
Implementation: rule `RA_Shares_balances_update_correctness`

6.5.2 Shares Debt Token
Reports: Debt tokens variable changes, Debt tokens high level props

6.5.2.1 receiveAllowances should change only on setReceiveApproval, decreaseReceiveAllowance, increaseReceiveAllowance, transferFrom.
Implementation: rule VC_SharesDebt_receiveAllowances_change
6.5.2.2 receiveAllowances should increase only on setReceiveApproval, increaseReceiveAllowance.
Implementation: rule VC_SharesDebt_receiveAllowances_increase
6.5.2.3 receiveAllowances should decrease only on setReceiveApproval, decreaseReceiveAllowance, transferFrom.
Implementation: rule VC_SharesDebt_receiveAllowances_decrease
6.5.2.4 Additive decreaseReceiveAllowance. receiveAllowances msg.sender after decreaseReceiveAllowance(amount) should be the same as decreaseReceiveAllowance(amount/2) + decreaseReceiveAllowance(amount/2).
Implementation: rule HLP_SharesDebt_additive_decreaseReceiveAllowance
6.5.2.5 Additive increaseReceiveAllowance. receiveAllowances msg.sender after increaseReceiveAllowance(amount) should be the same as increaseReceiveAllowance(amount/2) + increaseReceiveAllowance(amount/2).
Implementation: rule HLP_SharesDebt_additive_increaseAllowance
6.5.2.6 Integrity of setReceiveApproval. receiveAllowances of msg.sender after setReceiveApproval(amount) should be the exact amount that has been requested for a setReceiveApproval.
Implementation: rule HLP_SharesDebt_integrity_setReceiveApproval
6.5.2.7 Integrity of decreaseReceiveAllowance. receiveAllowances of msg.sender after decreaseReceiveAllowance(amount) should be equal to the receiveAllowances of the sender before request - amount.
Implementation: rule HLP_SharesDebt_integrity_decreaseReceiveAllowance
6.5.2.8 Integrity of increaseReceiveAllowance. receiveAllowances of msg.sender after increaseReceiveAllowance(amount) should be equal to the receiveAllowances of the sender before request + amount or uint256.max.
Implementation: rule HLP_SharesDebt_integrity_increaseReceiveAllowance

6.6 Silo

6.6.1 High Level Properties
Reports: Silo high level properties - DebtToken, Silo high level properties - CollateralOnlyToken, Silo high level properties - CollateralToken, Silo high level properties - Common

6.6.1.1 Inverse deposit - withdraw for collateralToken. For any user, the balance before deposit should be equal to the balance after depositing and then withdrawing the same amount.
Implementation: rule HLP_inverse_deposit_withdraw_collateral
6.6.1.2 Inverse deposit - withdrawFor for collateralToken. For any user, the balance before deposit should be equal to the balance after depositing and then withdrawing the same amount.
Implementation: rule HLP_inverse_deposit_withdrawFor_collateral
6.6.1.3 Inverse depositFor - withdraw for collateralToken. For any user, the balance before deposit should be equal to the balance after depositing and then withdrawing the same amount.
Implementation: rule HLP_inverse_depositFor_withdraw_collateral
6.6.1.4 Inverse depositFor - withdrawFor for collateralToken. For any user, the balance before deposit should be equal to the balance after depositing and then withdrawing the same amount.
Implementation: rule HLP_inverse_depositFor_withdrawFor_collateral
6.6.1.5 Inverse deposit - withdraw for collateralOnlyToken. For any user, the balance before deposit should be equal to the balance after depositing and then withdrawing the same amount.
Implementation: rule HLP_inverse_deposit_withdraw_collateralOnly
6.6.1.6 Inverse deposit - withdrawFor for collateralOnlyToken. For any user, the balance before deposit should be equal to the balance after depositing and then withdrawing the same amount.
Implementation: rule HLP_inverse_deposit_withdrawFor_collateralOnly
6.6.1.7 Inverse depositFor - withdraw for collateralOnlyToken. For any user, the balance before deposit should be equal to the balance after depositing and then withdrawing the same amount.
Implementation: rule HLP_inverse_depositFor_withdraw_collateralOnly
6.6.1.8 Inverse depositFor - withdrawFor for collateralOnlyToken. For any user, the balance before deposit should be equal to the balance after depositing and then withdrawing the same amount.
Implementation: rule HLP_inverse_depositFor_withdrawFor_collateralOnly

6.6.1.9 Inverse borrow - repay for debtToken. For any user, the balance before borrowing should be equal to the balance after borrowing and then repaying the same amount.
Implementation: rule HLP_inverse_borrow_repay_debtToken
6.6.1.10 Inverse borrow - repayFor for debtToken. For any user, the balance before borrowing should be equal to the balance after borrowing and then repaying the same amount.
Implementation: rule HLP_inverse_borrow_repayFor_debtToken
6.6.1.11 Inverse borrowFor - repay for debtToken. For any user, the balance before borrowing should be equal to the balance after borrowing and then repaying the same amount.
Implementation: rule HLP_inverse_borrowFor_repay_debtToken
6.6.1.12 Inverse borrowFor - repayFor for debtToken. For any user, the balance before borrowing should be equal to the balance after borrowing and then repaying the same amount.
Implementation: rule HLP_inverse_borrowFor_repayFor_debtToken
6.6.1.13 Additive deposit for collateralToken, totalDeposits while do deposit(x + y) should be the same as deposit(x) + deposit(y).
Implementation: rule HLP_additive_deposit_collateral
6.6.1.14 Additive deposit for collateralOnlyToken, collateralOnlyDeposits while do deposit(x + y) should be the same as deposit(x) + deposit(y).
Implementation: rule HLP_additive_deposit_collateralOnly
6.6.1.15 Additive depositFor for collateralToken, totalDeposits while do depositFor(x + y) should be the same as depositFor(x) + depositFor(y).
Implementation: rule HLP_additive_depositFor_collateral
6.6.1.16 Additive depositFor for collateralOnlyToken, collateralOnlyDeposits while do depositFor(x + y) should be the same as depositFor(x) + depositFor(y).
Implementation: rule HLP_additive_depositFor_collateralOnly
6.6.1.17 Additive withdraw for collateralToken, totalDeposits while do withdraw(x + y) should be the same as withdraw(x) + withdraw(y).
Implementation: rule HLP_additive_withdraw_collateral
6.6.1.18 Additive withdraw for collateralOnlyToken, collateralOnlyDeposits while do withdraw(x + y) should be the same as withdraw(x) + withdraw(y).
Implementation: rule HLP_additive_withdraw_collateralOnly
6.6.1.19 Additive withdrawFor for collateralToken, totalDeposits while do withdrawFor(x + y) should be the same as withdrawFor(x) + withdrawFor(y).
Implementation: rule HLP_additive_withdrawFor_collateral

6.6.1.20 Additive withdrawFor for collateralOnlyToken, collateralOnlyDeposits while do withdrawFor(x + y) should be the same as withdrawFor(x) + withdrawFor(y).
Implementation: rule HLP_additive_withdrawFor_collateralOnly
6.6.1.21 Additive borrow for debtToken, totalBorrowAmount while do borrow(x + y) should be the same as borrow(x) + borrow(y).
Implementation: rule HLP_additive_borrow_debtToken
6.6.1.22 Additive borrowFor for debtToken, totalBorrowAmount while do borrowFor(x + y) should be the same as borrowFor(x) + borrowFor(y).
Implementation: rule HLP_additive_borrowFor_debtToken
6.6.1.23 Additive repay for debtToken, totalBorrowAmount while do repay(x + y) should be the same as repay(x) + repay(y).
Implementation: rule HLP_additive_repay_debtToken
6.6.1.24 Additive repayFor for debtToken, totalBorrowAmount while do repayFor(x + y) should be the same as repayFor(x) + repayFor(y).
Implementation: rule HLP_additive_repayFor_debtToken
6.6.1.25 Integrity of deposit for collateralToken, totalDeposits after deposit should be equal to the totalDeposits before deposit + amount of the deposit.
Implementation: rule HLP_integrity_deposit_collateral
6.6.1.26 Integrity of deposit for collateralTokenOnly, collateralOnlyDeposits after deposit should be equal to the collateralOnlyDeposits before deposit + amount of the deposit.
Implementation: rule HLP_integrity_deposit_collateralOnly
6.6.1.27 Integrity of depositFor for collateralToken, totalDeposits after deposit should be equal to the totalDeposits before deposit + amount of the deposit.
Implementation: rule HLP_integrity_depositFor_collateral
6.6.1.28 Integrity of depositFor for collateralOnlyToken, collateralOnlyDeposits after deposit should be equal to the collateralOnlyDeposits before deposit + amount of the deposit.
Implementation: rule HLP_integrity_depositFor_collateralOnly
6.6.1.29 Integrity of withdraw for collateralToken, totalDeposits after withdrawal should be equal to the totalDeposits before withdrawal - the amount of the withdrawal.
Implementation: rule HLP_integrity_withdraw_collateral
6.6.1.30 Integrity of withdraw for collateralOnlyToken, collateralOnlyDeposits after withdrawal should be equal to the collateralOnlyDeposits before withdrawal - the amount of the withdrawal.
Implementation: rule HLP_integrity_withdraw_collateralOnly

6.6.1.31 Integrity of withdrawFor for collateralToken, totalDeposits withdrawal should be equal to the totalDeposits before withdrawal - the amount of the withdrawal.
Implementation: rule HLP_integrity_withdrawFor_collateral
6.6.1.32 Integrity of withdrawFor for collateralOnlyToken, collateralOnlyDeposits after withdrawal should be equal to the collateralOnlyDeposits before withdrawal - the amount of the withdrawal.
Implementation: rule HLP_integrity_withdrawFor_collateralOnly
6.6.1.33 Integrity of borrow for debtToken, totalBorrowAmount after borrow should be equal to the totalBorrowAmount before borrow + borrowed amount.
Implementation: rule HLP_integrity_borrow_debtToken
6.6.1.34 Integrity of borrowFor for debtToken, totalBorrowAmount after borrowFor should be equal to the totalBorrowAmount before borrowFor + borrowed amount.
Implementation: rule HLP_integrity_borrowFor_debtToken
6.6.1.35 Integrity of repay for debtToken, totalBorrowAmount after repay should be equal to the totalBorrowAmount before repay + repaid amount.
Implementation: rule HLP_integrity_repay_debtToken
6.6.1.36 Integrity of repayFor for debtToken, totalBorrowAmount after repayFor should be equal to the totalBorrowAmount before repayFor + repaid amount.
Implementation: rule HLP_integrity_repayFor_debtToken
6.6.1.37 Deposit of the collateral will only update the balance of msg.sender.
Implementation: rule HLP_deposit_collateral_update_only_sender
6.6.1.38 Deposit of the collateralOnly will only update the balance of msg.sender.
Implementation: rule HLP_deposit_collateralOnly_update_only_sender
6.6.1.39 DepositFor of the collateral will only update the balance of _depositor.
Implementation: rule HLP_depositFor_collateral_update_only_depositor
6.6.1.40 DepositFor of the collateralOnly will only update the balance of _depositor.
Implementation: rule HLP_depositFor_collateralOnly_update_only_depositor
6.6.1.41 Withdrawing of the collateral will only update the balance of msg.sender.
Implementation: rule HLP_withdraw_collateral_update_only_sender
6.6.1.42 Withdrawing of the collateralOnly will only update the balance of msg.sender.
Implementation: rule HLP_withdraw_collateralOnly_update_only_sender
6.6.1.43 WithdrawFor of the collateral will only update the balance of _depositor.
Implementation: rule HLP_withdrawFor_collateral_update_only_depositor
6.6.1.44 WithdrawFor of the collateralOnly will only update the balance of _depositor.
Implementation: rule HLP_withdrawFor_collateralOnly_update_only_depositor

6.6.1.45 Borrow will only update the balance of the msg.sender for debtToken.
Implementation: rule HLP_borrow_update_only_sender
6.6.1.46 BorrowFor will only update the balance of the borrower for debtToken.
Implementation: rule HLP_borrowFor_update_only_borrower
6.6.1.47 Repay will only update the balance of the msg.sender for debtToken.
I*mplementation: rule HLP_repay_update_only_sender*
6.6.1.48 RepayFor will only update the balance of the borrower for debtToken.
Implementation: rule HLP_repayFor_update_only_borrower
6.6.1.49 FlashLiquidate will only update the balances of the provided users. isSolventBefore == false => Balance for CollateralOnlyToken, CollateralToken should be 0.
Implementation: rule HLP_flashliquidate_shares_tokens_bal_zero

6.6.2 Risk Assessment
Reports: RA_Silo_no_double_withdraw, RA_Silo_no_negative_interest_for_loan, RA_Silo_balance_more_than_collateralOnly_deposit, RA_Silo_withdraw_all_shares, RA_Silo_borrowed_asset_not_depositable, RA_Silo_repay_all_shares, RA_Silo_repay_all_collateral

6.6.2.1 A user cannot withdraw the same balance twice (double spending).
Implementation: rule RA_Silo_no_double_withdraw
6.6.2.2 A user should not be able to repay a loan with less amount than he borrowed.
Implementation: rule RA_Silo_no_negative_interest_for_loan
6.6.2.3 With collateralOnly deposit, there is no scenario when the balance of a contract is less than that deposit amount.
Implementation: rule RA_Silo_balance_more_than_collateralOnly_deposit
6.6.2.4 A user should not be able to deposit an asset that he borrowed in the Silo.
Implementation: rule RA_Silo_borrowed_asset_not_depositable
6.6.2.5 A user has no debt after being repaid with max_uint256 amount.
Implementation: rule RA_Silo_repay_all_shares
6.6.2.6 A user can withdraw all with max_uint256 amount.
Implementation: rule RA_Silo_withdraw_all_shares

6.6.3 State Transition
Reports: Silo state transition - ST_Silo_asset_init_shares_tokes, Silo state transition - ST_Silo_asset_reactivate, Silo state transition - ST_Silo_mint_debt, Silo state transition - ST_Silo_totalSupply_collateralOnlyDeposits, Silo state transition - ST_Silo_totalSupply_totalBorrowAmount, Silo state transition - ST_Silo_mint_shares, Silo state transition - ST_Silo_totalSupply_totalDeposits

6.6.3.1 CollateralToken.totalSupply is changed => totalDeposits is changed.
Implementation: rule ST_Silo_totalSupply_totalDeposits
6.6.3.2 CollateralOnlyToken.totalSupply is changed => collateralOnlyDeposits is changed.
Implementation: rule ST_Silo_totalSupply_collateralOnlyDeposits
6.6.3.3 DebtToken.totalSupply is changed => totalBorrowAmount is changed.
Implementation: rule ST_Silo_totalSupply_totalBorrowAmount
6.6.3.4 AssetInterestData.interestRateTimestamp is changed and it was not 0 and AssetInterestData.totalBorrowAmount was not 0 => AssetInterestData.totalBorrowAmount is changed.
Implementation: rule ST_Silo_interestRateTimestamp_totalBorrowAmount_dependency
6.6.3.5 AssetInterestData.interestRateTimestamp is changed and it was not 0 and siloRepository.protocolShareFee() was not 0 => AssetInterestData.totalDeposits and AssetInterestData.protocolFees also changed.
Implementation: rule ST_Silo_interestRateTimestamp_fee_dependency
6.6.3.6 CollateralToken.totalSupply or collateralOnlyToken.totalSupply increased => deposit amount is not zero and asset is active.
Implementation: rule ST_Silo_mint_shares
6.6.3.7 DebtToken.totalSupply increased => borrow amount is not zero and asset is active.
Implementation: rule ST_Silo_mint_debt
6.6.3.8 AssetInterestData.status is changed to active and AssetStorage.collateralToken and AssetStorage.collateralOnlyToken and AssetStorage.debtToken where empty => AssetStorage.collateralToken and AssetStorage.collateralOnlyToken and AssetStorage.debtToken should not be empty and different.
Implementation: rule ST_Silo_asset_init_shares_tokes
6.6.3.9 AssetInterestData.status is changed to active and AssetStorage.collateralToken and AssetStorage.collateralOnlyToken and AssetStorage.debtToken where not empty => AssetStorage.collateralToken and AssetStorage.collateralOnlyToken and AssetStorage.debtToken should not update.
Implementation: rule ST_Silo_asset_reactivate

6.6.4 Valid States

Reports: Silo valid states

6.6.4.1 TotalDeposits is zero <=> collateralToken.totalSupply is zero.
Implementation: rule VS_Silo_totalDeposits_totalSupply
6.6.4.2 CollateralOnlyDeposits is zero <=> collateralOnlyToken.totalSupply is zero.
Implementation: rule VS_Silo_collateralOnlyDeposits_totalSupply
6.6.4.3 TotalBorrowAmount is zero <=> debtToken.totalSupply is zero.
Implementation: rule VS_Silo_totalBorrowAmount_totalSupply
6.6.4.4 AssetInterestData.lastTimestamp is zero => AssetInterestData.protocolFees is zero.
Implementation: rule VS_Silo_lastTimestamp_protocolFees
6.6.4.5 AssetInterestData.protocolFees increased => AssetInterestData.lastTimestamp and AssetStorage.totalDeposits are increased too.
Implementation: rule VS_Silo_protocolFees
6.6.4.6 AssetInterestData.totalBorrowAmount is not zero => AssetStorage.totalDeposits is not zero.
Implementation: rule VS_Silo_totalBorrowAmount
6.6.4.7 AssetInterestData.protocolFees is zero => AssetInterestData.harvestedProtocolFees is zero.
Implementation: rule VS_Silo_lastTimestamp_protocolFees_zero
6.6.4.8 AssetInterestData.status is active => AssetStorage.collateralToken is not empty and AssetStorage.collateralOnlyToken is not empty and AssetStorage.debtToken is not empty and allSiloAssets.length > 0.
Implementation: rule VS_Silo_active_asset

6.6.5 Variable Changes

Reports: Silo variable changes - VariableChanges, Silo variable changes - VariableChangesWithoutInterest, Silo variable changes - VariableChangesDebtToken, Silo variable changes - VariableChangesCollateralOnlyToken, Silo variable changes - VariableChangesCollateralToken

6.6.5.1 AssetStorage.totalDeposits can only change on deposit, depositFor, withdraw, withdrawFor, flashLiquidate, repay, repayFor, borrow, borrowFor, accrueInterest.
Implementation: rule VC_Silo_totalDeposits

6.6.5.2 AssetStorage.totalDeposits without _accrueInterest can only change on deposit, depositFor, withdraw, withdrawFor, flashLiquidate.
Implementation: rule VC_Silo_totalDeposits_without_interest
6.6.5.3 AssetStorage.collateralOnlyDeposits can only change on deposit, depositFor, withdraw, withdrawFor, flashLiquidate.
Implementation: rule VC_Silo_collateralOnlyDeposits
6.6.5.4 AssetStorage.totalBorrowAmount can only change on deposit, depositFor, withdraw, withdrawFor, flashLiquidate, repay, repayFor, borrow, borrowFor, accrueInterest.
Implementation: rule VC_Silo_totalBorrowAmount
6.6.5.5 AssetStorage.totalBorrowAmount without _accrueInterest can only change on deposit, depositFor, withdraw, withdrawFor.
Implementation: rule VC_Silo_totalBorrowAmount_without_interest
6.6.5.6 AssetInterestData.harvestedProtocolFees can only change on harvestProtocolFees.
Implementation: rule VC_Silo_harvestedProtocolFees
6.6.5.7 AssetInterestData.protocolFees can only change on deposit, depositFor, withdraw, withdrawFor, flashLiquidate, repay, repayFor, borrow, borrowFor, accrueInterest.
Implementation: rule VC_Silo_protocolFees
6.6.5.8 AssetInterestData.protocolFees without _accrueInterest can only change on borrow, borrowFor.
Implementation: rule VC_Silo_protocolFees_without_interest
6.6.5.9 AssetInterestData.interestRateTimestamp can only change on deposit, depositFor, withdraw, withdrawFor, flashLiquidate, repay, repayFor, borrow, borrowFor, accrueInterest.
Implementation: rule VC_Silo_interestRateTimestamp
6.6.5.10 AssetInterestData.interestRateTimestamp should not change in the same block.
Implementation: rule VC_Silo_interestRateTimestamp_in_the_same_block
6.6.5.11 AssetInterestData.status can only change on initAssetsTokens, syncBridgeAssets.
Implementation: rule VC_Silo_asset_status
6.6.5.12 AssetStorage.collateralToken and AssetStorage.collateralOnlyToken and AssetStorage.debtToken can only change on initAssetsTokens, syncBridgeAssets.
Implementation: rule VC_Silo_shares_tokens_change

6.6.5.13 CollateralToken.totalSupply can only change on deposit, depositFor, withdraw, withdrawFor, flashLiquidate.
Implementation: rule VC_Silo_collateral_totalSupply_change
6.6.5.14 CollateralOnlyToken.totalSupply can only change on deposit, depositFor, withdraw, withdrawFor if _collateralOnly is true and on flashLiquidate.
Implementation: rule VC_Silo_collateralOnly_totalSupply_change
6.6.5.15 DebtToken.totalSupply can only change on borrow, borrowFor, repay, repayFor.
Implementation: rule VC_Silo_debt_totalSupply_change
6.6.5.16 CollateralToken.totalSupply and AssetStorage.totalDeposits should increase only on deposit, depositFor.
Implementation: rule VC_Silo_collateral_totalDeposits_increase
6.6.5.17 CollateralOnlyToken.totalSupply and AssetStorage.collateralOnlyDeposits should increase only on deposit, depositFor if _collateralOnly is true.
Implementation: rule VC_Silo_collateralOnly_collateralOnlyDeposits_increase
6.6.5.18 CollateralToken.totalSupply and AssetStorage.totalDeposits should decrease only on withdraw, withdrawFor, flashLiquidate.
Implementation: rule VC_Silo_collateral_totalDeposits_decrease
6.6.5.19 CollateralOnlyToken.totalSupply and AssetStorage.collateralOnlyDeposits should decrease only on withdraw, withdrawFor if _collateralOnly is true and on flashLiquidate.
Implementation: rule VC_Silo_collateralOnly_collateralOnlyDeposits_decrease
6.6.5.20 DebtToken.totalSupply and AssetStorage.totalBorrowAmount should increase only on borrow, borrowFor.
Implementation: rule VC_Silo_debt_totalBorrow_increase
6.6.5.21 DebtToken.totalSupply and AssetStorage.totalBorrowAmount should decrease only on repay, repayFor.
Implementation: rule VC_Silo_debt_totalBorrow_decrease
6.6.5.22 AssetInterestData.interestRateTimestamp should only increase.
Implementation: rule VC_Silo_interestRateTimestamp_increase
6.6.5.23 The silo balance for a particular asset should only increase on deposit, depositFor,
repay, repayFor. The silo balance for a particular asset should only decrease on withdraw, withdrawFor, borrow, borrowFor, flashLiquidate, harvestProtocolFees.
Implementation: rule VC_Silo_balance

6. Verifications *(cont.)*

6.7 Silo Factory

Reports: Silo factory

6.7.1 siloRepository can only change on initRepository.\
Implementation: rule `VC_SiloFactory_siloRepository_change`
6.7.2 siloRepository can be initialized once. The second attempt should revert.\
Implementation: rule `HLP_SiloRepository_siloRepository_change`
6.7.3 Only the siloRepository can create a silo. \
Implementation: rule `UT_SiloRepository_createSilo_permissions`

6.8 Silo Repository

6.8.1 Valid States
Reports: Silo Repository - ValidStates

6.8.1.1 Solvency precision decimals are 10e18 and can not be changed.
Implementation: invariant VS_solvencyPrecisionDecimals
6.8.1.2 Default liquidation threshold ∈ (0, 10^18].
Implementation: invariant VS_defaultLiquidationThreshold
6.8.1.3 For every Silo and every asset assetConfig liquidation threshold ∈ (0, 10^18].
Implementation: invariant VS_siloLiquidationThreshold
6.8.1.4 Default max loan to value ∈ (0, 10^18].
Implementation: invariant VS_defaultMaxLTV
6.8.1.5 For every Silo and every asset assetConfig max loan to value ∈ (0, 10^18].
Implementation: invariant VS_siloMaxLTV
6.8.1.6 Default liquidation threshold is greater than default max loan to value.
Implementation: invariant VS_defaultLiquidationThresholdGreaterMaxLTV
6.8.1.7 For every Silo and every asset assetConfig liquidation threshold is greater than max loan to value.
Implementation: invariant VS_siloLiquidationThresholdGreaterMaxLTV
6.8.1.8 For every Silo and every asset assetConfig.liquidationThreshold == 0 <=> assetConfig.maxLoanToValue == 0.
Implementation: invariant VS_halfOfAssetConfigIsNeverEmpty
6.8.1.9 Entry fee ∈ (0, Solvency._PRECISION_DECIMALS].
Implementation: invariant VS_entryFee

6.8.1.10 Protocol share fee ∈ (0, Solvency._PRECISION_DECIMALS].
Implementation: invariant VS_protocolShareFee
6.8.1.11 Protocol liquidation fee ∈ (0, Solvency._PRECISION_DECIMALS].
Implementation: invariant VS_protocolLiquidationFee
6.8.1.12 Protocol liquidation fee ∈ (0, Solvency._PRECISION_DECIMALS].
Implementation: invariant VS_protocolLiquidationFee
6.8.1.13 Default Silo factory is never equal to zero address. If the factory version for an asset is not the default one, the Silo factory for this asset can be zero only if unregisterSiloVersion() is called. State after constructor call is not proved, but checked manually.
Implementation: rule VS_complexInvariant_siloFactory

6.8.2 Variable Changes
Reports: Silo Repository - VariableChanges

6.8.2.1 Default liquidation threshold can be set only by setDefaultLiquidationThreshold. ((Default liquidation threshold changed) <=> (f.selector == setDefaultLiquidationThreshold && msg.sender == owner)).
Implementation: rule VCH_setDefaultLiquidationThresholdOnlyOwner
6.8.2.2 Default max loan to value can be set only by setDefaultLiquidationThreshold. ((default max loan to value changed) <=> (f.selector == setDefaultMaximumLTV && msg.sender == owner)).
Implementation: rule VCH_setDefaultMaximumLTVOnlyOwner
6.8.2.3 Default interest rate model can be set only by setDefaultInterestRateModel. ((default max loan to value changed) <=> (f.selector == setDefaultInterestRateModel && msg.sender == owner)).
Implementation: rule VCH_setDefaultInterestRateModelOnlyOwner
6.8.2.4 Price providers repository can be set only by setPriceProvidersRepository. ((price provider repository changed) <=> (f.selector == setPriceProvidersRepository && msg.sender == owner)).
Implementation: rule VCH_setPriceProvidersRepositoryOnlyOwner
6.8.2.5 Router can be set only by setRouter. ((router changed) <=> (f.selector == setRouter && msg.sender == owner)).
Implementation: rule VCH_setRouterOnlyOwner
6.8.2.6 Notification receiver can be set only by setNotificationReceiver. ((notification receiver changed) <=> (f.selector == setNotificationReceiver && msg.sender == owner)).
Implementation: rule VCH_setNotificationReceiverOnlyOwner

6.8.2.7 Tokens factory can be set only by setTokensFactory. ((tokens factory changed) <=> (f.selector == setTokensFactory && msg.sender == owner)).
Implementation: rule VCH_setTokensFactoryOnlyOwner
6.8.2.8 Asset config updated <=> msg.sender is the owner.
Implementation: rule VCH_assetConfigOnlyOwner
6.8.2.9 ((new asset in getBridgeAssets()) <=> (f.selector == addBridgeAsset && msg.sender == owner)) && ((asset is removed from getBridgeAssets()) <=> (f.selector == removeBridgeAsset && msg.sender == owner)).
Implementation: rule VCH_bridgeAssets
6.8.2.10 ((new asset in getRemovedBridgeAssets()) <=> (f.selector == removeBridgeAsset && msg.sender == owner)) && ((asset is removed from getRemovedBridgeAssets()) <=> (f.selector == addBridgeAsset && msg.sender == owner)).
Implementation: rule VCH_removedBridgeAssets
6.8.2.11 When registerSiloVersion(..., isDefault) is called. msg.sender == owner && (latest version is default <=> isDefault == true).
Implementation: rule VCH_registerSiloVersionDefaultIsLatest
6.8.2.12 If the default Silo version is changed to newDefaultSiloVersion, then msg.sender == owner && (f.selector == registerSiloVersion(...,isDefault = true) || f.selector == setDefaultSiloVersion(..., siloVersion = newDefaultSiloVersion)).
Implementation: rule VCH_defaultSiloVersion

6.8.3 Unit Tests
Reports: Silo Repository - UnitTests

6.8.3.1 For every asset (getSilo(asset) == 0 || siloReverse(getSilo(asset)) == asset || getSilo(asset) == bridgePool()).
Implementation: invariant UT_getSiloReverseSilo
6.8.3.2 If the asset is a removed bridge asset, it is not a bridge asset.
Implementation: invariant UT_removedBridgeAssetIsNotBridge
6.8.3.3 If the asset is a bridge asset, it is not a removed bridge asset.
Implementation: invariant UT_bridgeAssetIsNotRemoved
6.8.3.4 Silo can be created for an asset in all cases, except (getSilo(asset) != 0 || assetIsABridge && (bridgeAssetsAmount == 1 || bridgePool != 0)). State after constructor call is not proved, but checked manually.
Implementation: invariant UT_complexInvariant_ensureCanCreateSiloFor

6.8.3.5 If the asset is a bridge asset, then Silo for this asset is not yet created or the Silo is a bridge pool.
Implementation: invariant UT_assetIsBridgeThenSiloIsBridgePool
6.8.3.6 If the asset is a removed bridge asset, then Silo for this asset is not yet created or the Silo is NOT a bridge pool.
Implementation: rule UT_assetIsBridgeThenSiloIsBridgePool

6.9 Tokens Factory

Reports: TokensFactory

6.9.1 _siloRepository can only change on initRepository.
Implementation: rule VC_TokensFactory_siloRepository_change
6.9.2 _siloRepository can be initialized once. The second attempt should revert.
Implementation: rule HLP_TokensFactory_siloRepository_change
6.9.3 createShareCollateralToken should revert if msg.sender != silo address.
Implementation: rule UT_TokensFactory_createShareCollateralToken_only_silo
6.9.4 createShareDebtToken should revert if msg.sender != silo address.
Implementation: rule UT_TokensFactory_createShareDebtToken_only_silo
6.9.5 _siloRepository can't be set to zero address if it was not zero.
Implementation: rule RA_TokensFactory_siloRepository_not_zero
6.9.6 Any silo should be able to create ShareCollateral and ShareDebt tokens.
Implementation: rule RA_TokensFactory_any_silo_can_create_shares

6.10 Guarded Launch

Reports: Guarded launch

6.10.1 maxLiquidity.globalLimit can only change on setLimitedMaxLiquidity() call. globalLimit changed => f.selector == setLimitedMaxLiquidity.
Implementation: rule VC_GuardedLaunch_globalLimit
6.10.2 maxLiquidity.defaultMaxLiquidity can only change on setDefaultSiloMaxDepositsLimit() call. defaultMaxLiquidity changed => f.selector == setDefaultSiloMaxDepositsLimit.
Implementation: rule VC_GuardedLaunch_defaultMaxLiquidity

6.10.3 For every Silo and it's every asset siloMaxLiquidity can only change on setSiloMaxDepositsLimit() call. SiloMaxLiquidity changed => f.selector == setSiloMaxDepositsLimit.
Implementation: rule VC_GuardedLaunch_siloMaxLiquidity
6.10.4 GlobalPause can only change on setGlobalPause() call. GlobalPause changed => f.selector == setGlobalPause.
Implementation: rule VC_GuardedLaunch_globalPause
6.10.5 For every Silo and it's asset siloPause can only change on setSiloPause() call. SiloPause changed => f.selector == setSiloPause.
Implementation: rule VC_GuardedLaunch_siloPause
6.10.6 Only owner can call setLimitedMaxLiquidity().
Implementation: rule UT_GuardedLaunch_setLimitedMaxLiquidity_onlyOwner
6.10.7 Only owner can call setDefaultSiloMaxDepositsLimit().
Implementation: rule UT_GuardedLaunch_setDefaultSiloMaxDepositsLimit_onlyOwner
6.10.8 Only owner can call setSiloMaxDepositsLimit().
Implementation: rule UT_GuardedLaunch_setSiloMaxDepositsLimit_onlyOwner
6.10.9 Only owner can call setGlobalPause().
Implementation: rule UT_GuardedLaunch_setGlobalPause_onlyOwner
6.10.10 Only owner can call setSiloPause().
Implementation: rule UT_GuardedLaunch_setSiloPause_onlyOwner
6.10.11 For any silo and any asset we must be sure that after it was paused we can unpause it.
Implementation: rule RA_GuardedLaunch_Silo_pause_unpause
6.10.12 If system been paused we must be sure that we can unpause it.
Implementation: rule RA_GuardedLaunch_Global_pause_unpause

6.11 Solvency

6.11.1 Unit Tests
Report: Unit tests

6.11.1.1 ConvertAmountsToValues return zero <=> amount * price < DECIMAL_POINTS.
Implementation: rule UT_convertAmountsToValues_zeroSanity

6.11.1.2 ConvertAmountsToValues returns array of the values, calculated as amount * price / DECIMAL_POINTS.
Implementation: rule UT_convertAmountsToValues_concreteFormula
6.11.1.3 CalculateLiquidationFee returns liquidationFeeAmount == amount * liquidationFee / DECIMAL_POINTS and newProtocolEarnedFees == protocolEarnedFees + liquidationFeeAmount. newProtocolEarnedFees is set to type(uint256).max value in case of the overflow.
Implementation: rule UT_calculateLiquidationFee
6.11.1.4 GetUserBorrowAmount returns user debt share balance to amount rounded up with compounded interest applied.
Implementation: rule UT_getUserBorrowAmount
6.11.1.5 GetUserCollateralAmount returns user collateral share balance to amount with compounded interest applied. Protocol interest is excluded from compounded interest.
Implementation: rule UT_getUserCollateralAmount
6.11.1.6 TotalBorrowAmountWithInterest returns totalBorrowAmount increased by compounded interest.
Implementation: rule UT_totalBorrowAmountWithInterest
6.11.1.7 TotalDepositsWithInterest returns totalDeposits increased by compounded interest. Protocol interest is excluded from compounded interest.
Implementation: rule UT_totalDepositsWithInterest