# Formal Verification *DRAFT v2* Report Blend v2

June 2025

*Prepared for*

**Script3 Ltd**

## Table of contents

# Project Summary

## Project Scope

| Project Name | Repository (link) | FV Commits | Platform |
|---|---|---|---|
| Blend v2 | https://github.com/blend-capital/blend-contracts-v2 | 996e09e | Stellar |

## Project Overview

This document describes the specification and verification of **Blend v2 Contracts** using the Certora Prover. The work was undertaken from **February 03, 2025,** to **March 13, 2025**.

The following contract files are included in our scope:

- `pool/src/*`

In addition, the `backstop` code was in scope for the ongoing contest Certora is organizing with Code4ena. This document does *not* include the work done by Certora in preparation for the contest.

The Certora Prover demonstrated that the implementation of the Stellar contracts above is correct with respect to the formal rules written by the Certora team. During the verification process, the Certora team discovered bugs in the Stellar contracts' code, as listed on the following page.

All our rules and config files are uploaded here: https://github.com/Certora/blend-contracts-v2/tree/certora:

- `confs` has all the config files for running the prover
- `spec` directory has all the rules we have written

They can be run like so: `certoraSorobanProver config_name.conf`. The sections below state additional changes and assumptions made for verification. For all contracts in scope, we report only on the properties listed below.

In addition, the team performed a manual audit of all the Stellar contracts. We share those findings in a separate report.

# Formal Verification

## Verification Notations

| | |
|---|---|
| Formally Verified | The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule. |
| Formally Verified After Fix | The rule was violated due to an issue in the code and was successfully verified after fixing the issue. |
| Violated | A counter-example exists that violates one of the assertions of the rule. |

## User Health Check Properties

Module General Assumptions

- Loop iterations: Any loop was unrolled at most 2 times (iterations)

User Health Property

**Summary.** We specified a user-health property to check that a user's health is checked after every action is submitted. Ideally, we would prove that a given user's health factor is always greater than some nominal value. However, due to the complexity of the arithmetic involved, this direct property is inherently difficult for automatic provers. Instead, we specify a weaker property: if a user's positions change, then either (i) it is in a way that *obviously* preserves the health factor, or (ii) a designated function is called to *check* that the user is still "healthy."

**Summarized code.** The code for handling each request type is complex. To make the verification feasible, we introduce *summaries* for functions exported by a given module `module_name` in the corresponding `pool/src/spec/summaries/<module_name>.rs`. We attempt to model the relevant behavior. It is important to note that not all of these summaries have been verified. Strictly speaking, we have not formally verified that they all soundly over-approximate the behavior of the summarized functions. However, all summaries are intended to be sound with respect to behaviors that affect the user health property (for example, we do not model writes to storage locations that are not needed for the property). Nevertheless, the correctness of the specification and verification results depends on the accuracy of these models.

The following is a high-level description of the functions summarized in each submodule of `pool`

- `events`
  Each function is summarized as simply a no-op, as we ignore emitting events.

- `submit`
  - `positions_hf_under`

    Set our model's flag saying that the designated "check" function has been called, and return an arbitrary boolean value.

  - `handle_transfers`, `handle_transfer_with_allowance`

Do nothing.

- `actions`
  - `apply_supply`
    Choose a reserve nondeterministically and *increase* the user's collateral of that reserve by an arbitrary positive value.
  - `apply_withdraw_collateral`
    Choose a reserve nondeterministically and *decrease* the user's collateral of that reserve by an arbitrary positive value.
  - `apply_borrow`
    Choose a reserve nondeterministically and *increase* the user's liability of that reserve by an arbitrary positive value.
  - `apply_repay`
    Choose a reserve nondeterministically and *decrease* the user's liability of that reserve by an arbitrary positive value.
  - `build_actions_from_request`
    Returns an arbitrary `Actions` such that either `check_health` is true (indicating that positions_hf_under should be called) or that the arbitrarily chosen User state's positions preserve the health of the original positions (e.g., simulating the effect of increasing collateral, decreasing liabilities, etc.).

- `auction`
  - `fill`
    Set the user's collateral and liabilities to new arbitrary values.
  - `delete`
    Do nothing.

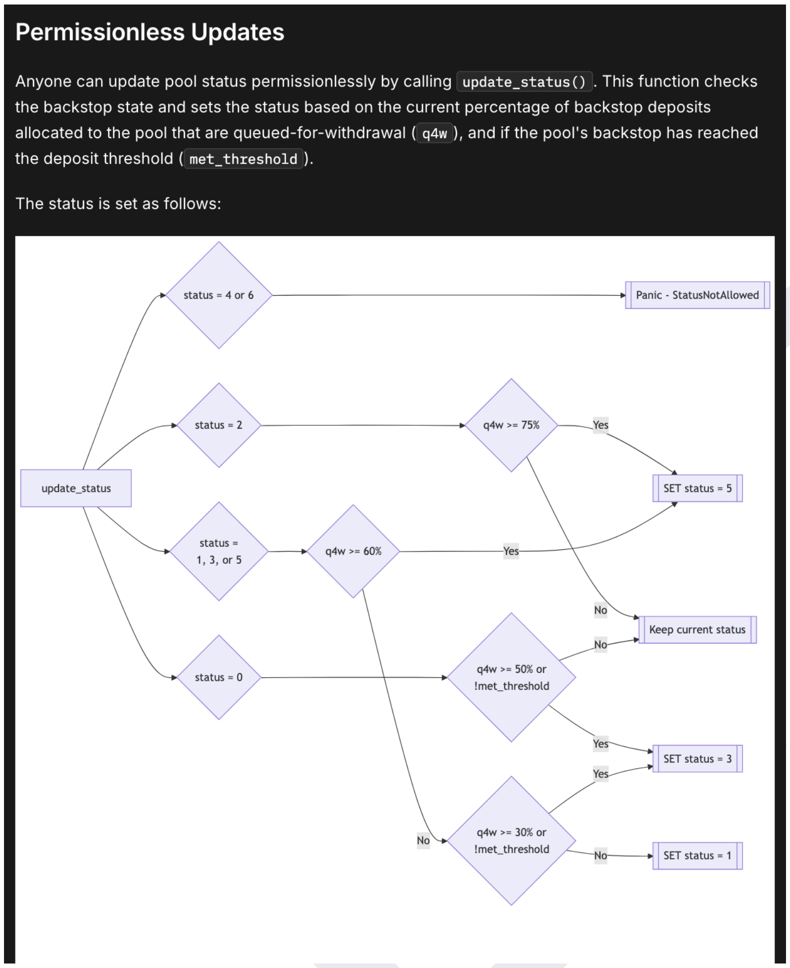
All the rules are in `pool/src/spec/health_rules.rs,`
and the config file is `pool/confs/health.conf`.

| User health is checked for submitted actions | |
|---|---|
| Status: Verified | |

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **build_actions_from_request** | Verified | Verifies the soundness of the summary used for `build_actions_from_request` | *Report* |
| **user_health_execute_submit** | Verified | Verifies the health property is maintained by `execute_submit` | *Report* |
| **user_health_execute_submit_with_flash_loan** | Verified | Verifies the health property is maintained by `execute_submit_with_flash_loan` | *Report* |
| **handle_transfer_with_allowance_summary_ok** | Verified | Verifies the soundness of the summary used for `handle_transfer_with_allowance` | *Report* |
| **handle_transfers_summary_ok** | Verified | Verifies the soundness of the summary used for `handle_transfers` | *Report* |

## Pool Status Properties

**Summary**. Based on the specification in the Blend documentation, we wrote state machine properties to make sure that `execute_update_pool_status` correctly updates the pool status as indicated in the diagram in the documentation. Diagram from documentation used as specification for pool status properties:



**Permissionless Updates**

Anyone can update pool status permissionlessly by calling `update_status()`. This function checks the backstop state and sets the status based on the current percentage of backstop deposits allocated to the pool that are queued-for-withdrawal (`q4w`), and if the pool's backstop has reached the deposit threshold (`met_threshold`).

The status is set as follows:

**Summarized code.** To verify these properties, we summarized some of the code that was not relevant to the property. We also use `#[cfg(feature = "certora")]` and `#[cfg(not(feature = "certora"))]` in `execute_update_pool_status` to conditionally compile code when the `certora` feature is enabled.

The summary we used is `pool_data_summary()` in `pool/src/spec/summaries/backstop_pooldata.rs`. This returns a non-deterministically chosen `PoolBackstopData` instead of getting one via the backstop client.

We also rely on two GHOST variables: `GHOST_MET_THRESHOLD`, `GHOST_POOL_BACKSTOP_DATA`:

- `GHOST_MET_THRESHOLD` tracks the value of `met_threshold` in `execute_update_pool_status`
- `GHOST_POOL_BACKSTOP_DATA` tracks the `PoolBackstopData`

All the rules are in `pool/src/spec/pool_status_rules.rs` and the config files are named as `pool/confs/pool_statusN.conf`.

## Pool status is correctly updated

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **verify_update_status_6** | Verified | When pool status is `6` (setup), always panic | Report |
| **verify_update_status_4** | Verified | When pool status is `4` (admin frozen), always panic | Report |
| **verify_update_status_2** | Verified | When pool status is `2` (admin frozen), change status to `5` (admin on-ice) if `q4w >= 0_7500000` | Report |
| **verify_update_status_0_a** | Verified | When pool status is `0` (admin active), set status to `3` (on ice) if threshold is not met OR `q4w >= 0_5000000` | Report |

| | | | |
|---|---|---|---|
| **verify_update_status_0_b** | Verified | When pool status is `0` (admin active), keep status `0` if `!(threshold is not met OR q4w >= 0_5000000)` | Report |
| **verify_status_update** | Verified | After `execute_update_pool_status`, status must be 0, 1, 2, 3, 5 | Report |
| **verify_update_status_other_a** | Verified | When pool status is not `0`, `2`, `4`, or `6`, and `q4w >= 60%`, set status to `5` | Report |
| **verify_update_status_other_b** | Verified | When pool status is not `0`, `2`, `4`, or `6`, and the threshold is not met or `q4w >= 30%`, set status to `3` | Report |
| **verify_update_status_other_c** | Verified | When pool status is not `0`, `2`, `4`, or `6`, and the above conditions are not true, set the status to `1` | Report |

## User Integrity Properties

**Summary**. We wrote the following rules that check integrity properties for various user operations. These check that the operations correctly update values like pool collateral, d_supply, etc.

**Summarized code.** To verify these properties, we summarized `update_emissions`. The summary is in `pool/src/spec/summaries/emissions.rs` and the summarized version is compiled using the certora feature: `#[cfg(feature = "certora")]`.

We *did not* formally prove the summary sound. Note that these summaries are only intended for the properties in this section. A more precise model of emissions will be required to prove properties that depend on emissions.

The summary has two components:

- `update_emission_data_summary`
  - This returns a `nondet ReserveEmissionData` if there is an entry in storage corresponding to `res_token_id`. It also updates the storage by associating this `nondet ReserveEmissionData` to the `res_token_id`.
  - This returns `None` if there is no entry in storage corresponding to `res_token_id`.
- `update_emissions_summary`
  - This calls `update_emission_data_summary` and changes user emission (`set_user_emissions`) by setting `claim` to be `false` and `accrued` to be a `nondet`.

## User operation integrity

| | | | |
|---|---|---|---|
| **add_liabilities_increases_liabilities** | Verified | Adding liabilities increases liabilities | *Report* |
| **add_liabilities_increases_dsupply** | Verified | Adding liabilities increases `d_supply` on reserve, unchanged `b_supply` | *Report* |
| **remove_liabilities_decreases_liabilities** | Verified | Removing liabilities decreases liabilities | *Report* |
| **remove_liabilities_decreases_dsupply** | Verified | Removing liabilities decreases `d_supply` on reserve, unchanged `b_supply` | *Report* |
| **add_collateral_increases_position_collateral** | Verified | Adding collateral increases collateral | *Report* |
| **add_collateral_increases_b_supply** | Verified | Adding collateral increases `b_supply`, unchanged `d_supply` | *Report* |
| **remove_collateral_decreases_position_collateral** | Verified | Removing collateral decreases collateral | *Report* |
| **remove_collateral_decreases_b_supply** | Verified | Removing collateral decreases `b_supply`, unchanged `d_supply` | *Report* |
| **add_supply_increas** | Verified | Adding supply increases positional supply | *Report* |

| | | | |
|---|---|---|---|
| **es_position_supply** | | | |
| **add_supply_increases_b_supply** | Verified | Adding supply increases `b_supply`, unchanged `d_supply` | *Report* |
| **remove_supply_decreases_position_collateral** | Verified | Removing supply decreases positional supply | *Report* |
| **remove_supply_decreases_b_supply** | Verified | Removing collateral decreases `b_supply`, unchanged `d_supply` | *Report* |

# Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, 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.

Even though we hope this information is helpful, we 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.

# About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.