certora

# Security Assessment & Formal Verification Report

# Manifest

November 2024

Prepared for Manifest

certora

# Table of content

certora

# Project Summary

## Project Scope

| Project Name | Repository (link) | Audited Commits | Platform |
|---|---|---|---|
| Manifest | https://github.com/CKS-Systems/manifest | be0c627 | Solana |
| | | fd905fd<br>PR #108: sort by regular price only | Solana |

In addition to PR #108, we also cherry-picked commit 714be51, which removes the tail recursion from the `insert_fix` and `remove_fix` functions. This makes it easier to verify the correctness of those critical parts of the code.

## Project Overview

This document describes the specification and verification of **Manifest smart contracts** using the Certora Prover and manual code review findings. The work was undertaken from **28 August 2024** to **21 November 2024.**

The following contract list is included in our scope:

```
/programs/manifest/src/*
/lib/src/red_black_tree.rs
```

The Certora Prover demonstrated that the implementation of the **Solana** contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solana contracts**.** During the verification process and the manual audit, the Certora team discovered bugs in the Solana contracts code, as listed on the following page.

## Findings Summary

The table below summarizes the findings of the review, including type and severity details.

| Severity | Discovered | Confirmed | Fixed |
|---|---|---|---|
| Critical | - | - | - |
| High | 2 | 2 | 2 |
| Medium | 3 | 2 | 2 |
| Low | 7 | 4 | 4 |
| Informational | 14 | 10 | 10 |
| **Total** | 26 | 18 | 18 |

## Severity Matrix

| | | | | |
|---|---|---|---|---|
| **Impact** | High | Medium | High | Critical |
| | Medium | Low | Medium | High |
| | Low | Low | Low | Medium |
| | | Low | Medium | High |
| | | **Likelihood** | | |

certora

# Detailed Findings

| ID | Title | Severity | Status |
|---|---|---|---|
| H-01 | Withdraw transfers funds from user to vault instead of vault to user for token 2022 | High | Fixed in PR #173 |
| H-02 | hinted_cancel_index isn't checked to match block size during batch update | High | Fixed in PR #226 |
| M-01 | Swap function might swap for less amount out under low liquidity | Medium | Not confirmed |
| M-02 | expand_global() expands the account's memory by the wrong size | Medium | Fixed in PR #66 |
| M-03 | Swap function might return less than amount out or revert due to unbacked global orders | Medium | Fixed in PR #119 |
| L-01 | L-01 Refund for order cancellation is rounded down rather than up (PR #108) | Low | Fixed in PR #140 |
| L-02 | Attacker can remove global order by partially fulfilling on another market | Low | Not Confirmed |
| L-03 | Attacker can front run an order with posting multiple small orders, gaining profit from | Low | Not Confirmed |

| | | | |
|---|---|---|---|
| | rounding | | |
| L-04 | Attacker can front run global or post only order with a small order to cause it to revert | Low | Not Confirmed |
| L-05 | Sell orders can leave rounding errors in the contract forever | Low | Fixed in PR #39 |
| L-06 | Use of floating point numbers f64 can cause unintended consequences | Low | Fixed |
| L-07 | Red Black tree marks wrong color during removal, leading to an unbalanced/inefficient tree | Low | Fixed in PR #193 |

## Featured Finding: Avoid complications of Effective Price by rounding in favor of large orders

Following is a feedback we got from Manifest's team as a response to our initial report:

*"We feel that the recommendation to change our rounding from effective price to taker rounding was not prominently featured. We feel that was the biggest contribution of the audit.*

*We expected that finding/recommendation to be much more highlighted since it radically simplified the logic and we suspect it simplifies requirements of formal verification too."*

Thank you for the kind words. Following your comments, we decided to add a new issue (I-11 - Avoid complications of Effective Price by rounding in favor of large orders) where we provide more details about that recommendation and mention it here at the beginning of our report.

## High Severity Issues

### H-01 Withdraw transfers funds from user to vault instead of vault to user for Token-2022

| Severity: **High** | Impact: **High** | Likelihood: **High** |
| --- | --- | --- |
| Files: programs/manifest/src/program/processor/withdraw.rs | Status: Fixed | Violated Properties: No loss of funds and Integrity of withdraw<br>rule_withdraw_base<br>rule_withdraw_quote<br>rule_withdraw_withdraws |

**Description:** The withdraw instruction allows the user to withdraw their funds from the vault to their wallet.
However, for spl token 2022, the function transfers the funds from the user to the vault rather than the vault to the user. Thus, instead of receiving funds, the user would be charged the same amount.

```
Unset
&spl_token_2022::instruction::transfer_checked(
                token_program.key,
                trader_token.key,
                if is_base {
                    dynamic_account.fixed.get_base_mint()
                } else {
                    dynamic_account.get_quote_mint()
                },
                vault.key,
                payer.key,
                &[],
                amount_atoms,
                if is_base {
                    dynamic_account.fixed.get_base_mint_decimals()
```

```
        } else {
            dynamic_account.fixed.get_quote_mint_decimals()
        },
    )?,
```

**Exploit Scenario:**

- Bob deposits 10K USDC to the protocol
- Bob trades it for 300K FLUXB
- Bob attempts to withdraw the funds, but instead of withdrawing the protocol transfers an additional 300K FLUXB from them to the protocol
- Bob has lost 600K FLUXB in total, with no way to rescue the funds

**Recommendations:** Correct the transfer to transfer the funds from the vault to the user

**Customer's response:** Fixed in PR #173

**Fix Review:** Fix confirmed

## H-02 `hinted_cancel_index` isn't checked to match block size during batch update

| Severity: **High** | Impact: **High** | Likelihood: **Medium** |
|---|---|---|
| Files: programs/manifest/src/program/processor/batch_update.rs | Status: Fixed | |

**Description:** As part of batch update, the user can pass a list of orders to cancel.
The user can refer to the order by passing the location of the index in memory as the `hinted_cancel_index` parameter, the function then intends to verify that the index matches the alignment of the account's memory (by checking that it's a multiple of `MARKET_BLOCK_SIZE`).
However, the function checks the wrong parameter - it checks `trader_index` rather than `hinted_cancel_index`.
A user might be passing the wrong index, causing unintended consequences to the account's memory and paying the user for canceling an order he never posted.

```
C/C++
    require!(
        trader_index % (MARKET_BLOCK_SIZE as DataIndex) == 0,
        ManifestError::WrongIndexHintParams,
        "Invalid cancel hint index {}",
        hinted_cancel_index,
    )?;
```

**Recommendations:** Change `trader_index` to `hinted_cancel_index`

**Customer's response:** Fixed in PR #226.
*We would like the report to show an adversarial constructed example though. That would illustrate it better, an example that goes through and shows how canceling with a wrong hint can lead to an attacker trader index receiving funds.*

**Fix Review:** Fix confirmed

## Medium Severity Issues

| M-01 Swap function might swap for less amount out under low liquidity | | |
|---|---|---|
| Severity: **Medium** | Impact: **High** | Likelihood: **Low** |
| Files: programs/manifest/src/program/processor/swap.rs | Status: Not Confirmed | |

**Description:** The swap instruction calls `place_order()` and passes `base_atoms` as the amount of base atoms that should be traded.
However, it doesn't check that the full amount has been traded. In case there's not enough liquidity in the current orders the function would return after a partial fulfillment.
In case of swapping base for quote where `is_exact_in=false` the user might end up with less amount out for the full amount in

**Exploit Scenario:**

- Eve creates a bid order with the price of 1 and 1000 quote atoms
- Eve creates another order with the price of 0.1 and 100 quote atoms
- Bob tried to swap a 1000 base for a 1000 quote
    - Bob sets `is_exact_in=false`
- Eve front runs this and removes the order with the price of 1
- Bob's tx is executed, the swap instruction swaps 1000 for 100
    - `impact_base_atoms()` would set the `base_atoms` to 10,000, but the `place_order()` would trade only 1000
- Bob lost 900 base atoms

**Recommendations:** Check the `base_atoms_traded` return value of the `place_order()` function to verify that the full amount has been traded.

**Customer's response:** Working as intended. We added a comment to the code affirming that this is intended behavior at PR #121

**Fix Review:** The new comments clarify the design, raise awareness of the risk, and encourage the caller program to do additional checks to avoid the described issue.

### M-02 `expand_global()` expands the account's memory by the wrong size

| Severity: **Medium** | Impact: **Low** | Likelihood: **High** |
|---|---|---|
| Files: programs/manifest/src/program/processor/shared.rs | Status: Fixed | |

**Description:** Before a new trader is added to the global account `expand_global()` is called to expand the memory of the account and make room for the new trader.
However, the function calls `expand_global_fixed()` which expands it by 80 bytes (the market's block size) rather than 64 bytes (the global account block size).
This increases the cost for the caller, and makes it run out of space faster - reducing the amount of traders that can fit in one account.

**Recommendations:** Expand by 64 bytes rather than 80

**Customer's response:** Fixed in PR #66.
This is very low impact. The number of global seats is capped, so it is very limited in the amount of impact (~1600 bytes extra allocated per global, once for the lifetime of the program).

**Fix Review:** Fix confirmed

### M-03 Swap function might return less than amount out or revert due to unbacked global orders

| Severity: **Medium** | Impact: **Medium** | Likelihood: **Medium** |
|---|---|---|
| Files: programs/manifest/src/program/processor/swap.rs | Status: Fixed | |

**Description:** The `impact_base_atoms()` function is supposed to estimate how many base atoms are needed to be sold (or bought) to receive (or in exchange for) some amount of quote atoms.
However, this function doesn't check if global orders have enough funds to back them. This leads to an underestimation (in case of ask) or overestimation (in case of bid) of the base atoms.
(the same bug exists on `impact_quote_atoms()`, but that isn't used by any other function)

This has 2 different impacts:

- In the case of swapping base for quote - the user would receive less than the amount out
- In case of swapping quote for base - the swap might revert even when the user's order can be fulfilled

**Exploit Scenario:**

Swapping base for quote:

- Eve creates a global bid order with a price of 1 and a regular bid order with a price of 0.5
- Eve withdraws the funds from the global account, so the global order doesn't have funds to back it
- Bob attempts to swap base for a quote
    - atoms in = 1000
    - atoms out = 1000
    - is exact in = false
    - is base in = true

- The impact function would match the global order, setting the base atoms to 1000
- The `place_order()` function would remove the global order since it has no funds to back it, it'd then trade the 1000 base for 500 quote atoms
- The swap would complete with only 500 atoms out, half the amount that the user has specified

Swapping quote for base:

- Alice creates a regular ask order with the price of 1
- Eve creates a global ask order with the price of 0.5
- Eve withdraws the funds from the global account, so the global order doesn't have funds to back it
- Bob attempts to swap the quote for a base
    - atoms in = 1000
    - atoms out = 1000
    - is exact in = true
    - is base in = false
- The impact function would match the global order, setting the base atoms to 2000
- The `place_order()` function would remove the global order since it has no funds to back it, it'd then attempt to trade the1000 quote for 2000 base atoms
    - This would cause the function to revert, due to either insufficient balance for Bob or due an underflow in the swap function
- This effectively DoS-es the swap function

**Recommendations:** Fix `impact_*_atoms()` functions to check if global orders have sufficient funds to back them

**Customer's response:** Fixed in PR #119.
Comments in code now clarify that the desired parameter to swap is only best effort.

**Fix Review:** Fix confirmed

## Low Severity Issues

### L-01 Refund for order cancellation is rounded down rather than up (PR #108)

| Severity: **Low** | Impact: **Low** | Likelihood: **High** |
|---|---|---|
| Files: programs/manifest/src/state/market.rs | Status: Fixed | Violated Property: rule_cancel_order_by_index_bid rule_cancel_order_bid |

**Description:** As part of the audit the team has reviewed PR #108 which removes the effective price.
The team has found that during cancellation of an order the refund amount is rounded down rather than up, this would cause a minor loss of funds (1 atom) for the trader.

**Recommendations:** Round up for the refund amount of a canceled order

**Customer's response:** Fixed in PR #140.
This is related to removal of effective price which was the biggest contribution of the audit.

**Fix Review:** Fix confirmed

## L-02 Attacker can remove global order by partially fulfilling on another market

| Severity: **Low** | Impact: **Low** | Likelihood: **Medium** |
|---|---|---|
| Files: programs/manifest/src/state/market.rs | Status: Not Confirmed | |

**Description:** Global orders are removed from the market if the funds to back them are less than the amount that the taker is trying to trade.
An attacker can use this to remove global orders that they consider as unwanted competition.

**Exploit Scenario:**

- Bob deposits 10K USDC to the USDC global account
- Bob places a 10K USDC order on the USDC/SOL market
- Bob also places 100 USDC order on the USDC/USDT market
- Eve wants to remove Bob's order from the USDC/SOL market (since his order has a better price, so takers match his order first)
- Eve fulfills 1 atom of Bob's order on the USDC/USDT market
- Eve then tries to fully match Bob's global order
    - Since Bob has 1 less atom than needed the order would be removed
    - Eve doesn't have to spend anything on this, as long as the next order has a slightly worse price the instruction would end at this point (Eve would specify Bob's price as the desired price)

**Recommendations:** Consider allowing partial fulfillment of global orders

**Customer's response:** This is working as intended. We want people to overprovision global. Likely not going to do the todo here . There is economic protection though since the attacker needs to pay for it. In other words, the attacker needs to put on risk that they need to hold and cannot necessarily get out of without significant cost. Another consideration is this increases market competitiveness - the less funds backing your orders the less competitive they are and

should be. The decision to do this is going to decide between the marginal benefit in edge cases we don't really want (underprovisioned global balance where the maker is stretched too thin), vs the likely consequences in user behavior for how often they attempt to grab the global lock. Currently leaning towards not addressing this edge case because we want to limit usage of global to where it is needed and not encourage extra competition for landing transactions.

### L-03 Attacker can front run an order with posting multiple small orders, gaining profit from rounding

| Severity: **Low** | Impact: **High** | Likelihood: **Very low** |
|---|---|---|
| Files: programs/manifest/src/state/market.rs | Status: Not Confirmed | |

**Description:** Under current implementation trades always round up in favor of the maker.
An attacker can take advantage of that, and front-run a takers tx, creating multiple small orders that would be fulfilled while rounding up in favor of the attacker. Causing a loss of up to 50% of the trade.

**Exploit Scenario:**

- Market USDC/USDT is created with USDC as quote and USDT as base
- Bob attempts to buy 100 USDT atoms with the price of 1.0001
- Eve front runs Bob and creates 100 small orders with 1 atom each and a price of 1.0001
- Since every order is rounded up, Bob ends up paying 200 USDT, almost double the price than he desired

**Recommendations:** It is possible to get rid of the complications that are caused by using effective_price. Instead of rounding in favor of the maker, the code should round in favor of the taker when he matches the full order and in favor of the maker when the order is only partially matched. This way, every user (taker and maker) is protected against the other party introducing micro-trades and would lose less than one atom on each of their trades. The effective price is no longer needed and orders can be sorted by price instead. Partially fulfilled orders can keep their place in the book. See I-11 for more information.

**Customer's response:** This was not possible because of effective price. That was the whole point of effective price.

**Fix Review:** Since PR #108 this issue does not exist anymore.

### L-04 Attacker can front run global or post only order with a small order to cause it to revert

| Severity: **Low** | Impact: **Low** | Likelihood: **Medium** |
|---|---|---|
| Files: programs/manifest/src/state/market.rs | Status: Not Confirmed | |

**Description:** When a user attempts to post a global or post-only order, another user can front-run them and post a very small order that would match it.

Since global and post orders revert if there's any match, the transaction would revert.

Note that this would work only if the global/post-only order is a better price than all other orders on the market. Otherwise, the attacker would have to fulfill some orders before they can post an order that would match the victim's order.

The incentive for the attacker might be preventing competition, in case they already have an order in the market and they want to prevent others from offering a better price than them.

**Exploit Scenario:**

- Eve has posted an ask order on the market with the price of 1
- Bob tries to post a global ask order with the price of 0.99
- Eve front runs it and creates  a bid order of 1 atom with the price of 0.995
- Bob tx would match the bid order and revert

**Recommendations:** Consider making global orders not post-only. As for post-only orders - maybe allow matching up to some dust amount (this can be a parameter that the user can pass).

**Customer's response:** This is part of the spec. We don't want to allow global to be used for taking since that would encourage much more lock contention on the global accounts. As for the post only case, that's the reason for post only instead of limit. Will not make any changes here. This is the same consideration from a transaction landing perspective as L-02.

### L-05 Sell orders can leave rounding errors in the contract forever

| Severity: **Low** | Impact: **Very Low** | Likelihood: **High** |
|---|---|---|
| Files: programs/manifest/src/state/market.rs | Status: Fixed | |

**Description:** When an order is posted, a balance is subtracted from the trader's balance and held in the order till a trade is made or the order is canceled/expired.
That balance is represented in the `num_base_atoms` field. For bid orders - since the order's balance is in terms of quote, then the balance is calculated by multiplying `num_base_atoms` by the order's price, rounded up (after the change in PR #108).
When a bid order is fulfilled partially, `num_base_atoms` is set to the amount that would represent the new order's balance.
However, this new value is rounded down, causing a minor (dust) loss of funds to the trader.

**Exploit Scenario:**

- Bob creates a bid order with the price of 0.1 and 75 base atoms
  - 8 quote tokens are removed from the Bob's balance: $\lceil 75 * 0.1 \rceil = 8$
- Alice fulfills the order with 26 base atoms
  - Quote traded are rounded down: $\lfloor 26 * 0.1 \rfloor = 2$
  - `num_base_atoms` is set to $(75 - 26) = 49$
- The order is canceled by Bob
  - Bob receives $\lceil 49 * 0.1 \rceil = 5$ quote tokens in return
- Bob was charged 8 quote tokens to create the order, 2 went to the trader and 5 were returned to Bob. 8-2-5=1, 1 quote token was lost in the process.

**Recommendations:** In case of partial fulfillment, refund the order owner for that difference.

**Customer's response:** Fixed in PR #39, All part of the effective price stuff.
**Fix Review:** Fix confirmed

### L-06 Use of floating point numbers f64 can cause unintended consequences

| Severity: **Low** | Impact: **Low** | Likelihood: **High** |
|---|---|---|
| Files: programs/manifest/src/quantities.rs | Status: Fixed | |

**Description:** Early version of the code used floating point numbers (f64) for price computation. There are several issues with floating point numbers. First, the SBF virtual machine does not support floats. Thus, the behavior of the floats on SBF might be different than expected during testing. Second, floats are not able to represent common decimal numbers. For example, in floats, 0.2 + 0.1 is not 0.3. Third, the rounding mode in floats is neither floor nor ceiling, making it difficult to implement algorithms where rounding is important. Finally, floats are very difficult for formal verification.

**Recommendations:** Use fixed point decimal instead of floats

**Customer's response:** An early version of the code used f64, but at the recommendation of Certora, we switched our internal representations of prices to decimal fixed point. Don't have a linked pull request since this is from before we made the repo public.

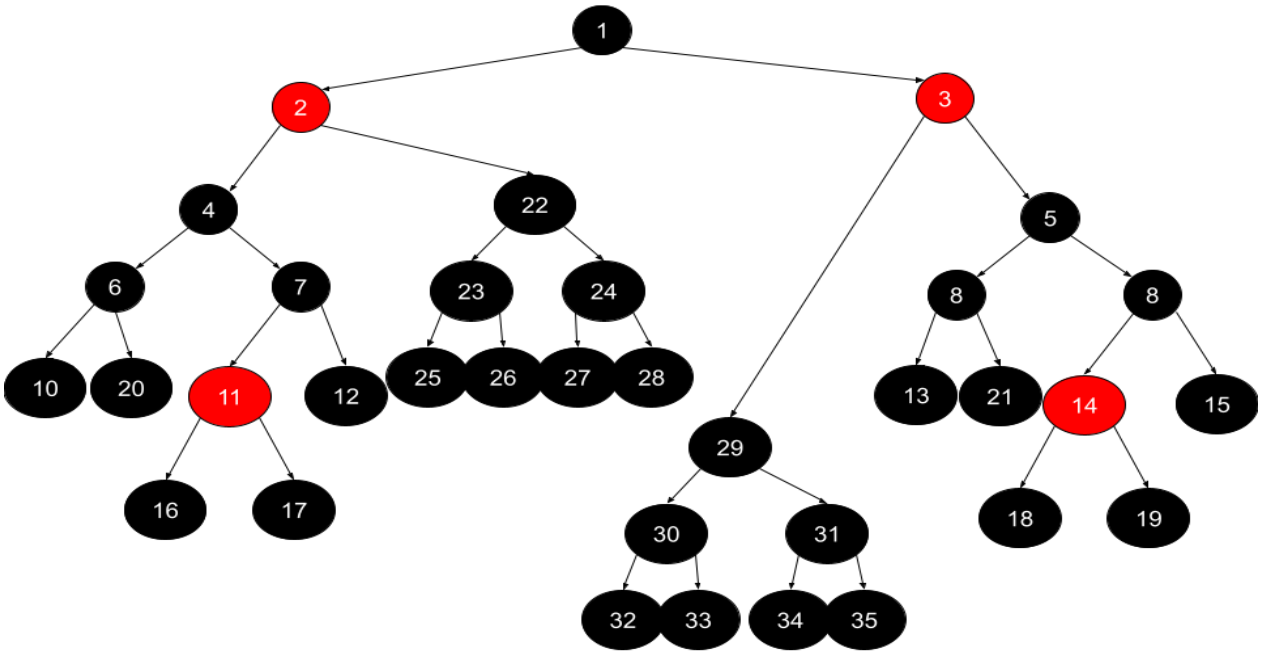
**Fix Review:** Fix confirmed

### L-07 Red Black tree marks wrong color during removal, leading to an unbalanced/inefficient tree

| Severity: **Low** | Impact: **Low** | Likelihood: **Medium** |
|---|---|---|
| Files: lib/src/red_black_tree.rs | Status: Fixed | Violated Property: rule_remove_fix_matches_reference_case3_left_child rule_remove_fix_matches_reference_case3_right_child |

**Description:** During removal of a node in the RB tree the `remove_fix()` is called to re-paint the nodes in the tree to the right color. In some cases the sibling's child index is painted red when it should be painted as the parent's color.
This leads to an unbalanced tree, consuming more computation units for future operations.

Example of an unbalanced tree created as a result of the bug: If in the following tree the node 6 is removed the tree ends up with node 2 and its left child being colored red.



**Recommendations:** Set the correct color - the parent's color rather than red

**Customer's response:** Fixed in PR #193.
Likelihood is very low. We ran fuzz testing and it didn't run into it. It is a very specific 35 node tree. Arguably, the impact is negligible since it is still a valid BST, just not balanced.

**Fix Review:** Fix confirmed

## Informational Issues

### I-01 During cancellation, `order_sequence_number` isn't checked, possibly emitting wrong sequence number

**Description:** in `batch_update()` if the user supplies an `order_index_hint` then `order_sequence_number` isn't checked.
This unchecked sequence number is then emitted as part of the event, possibly emitting the wrong sequence number.

**Recommendation:** Check that the sequence number is right (or set it to the correct value if it's not set by the user)

**Customer's response:** Fixed in PR #120

**Fix Review:** Fix confirmed

### I-02 RestingOrder trees lookup function is broken for nodes with same price

**Description:** Resting order equality trait implementation checks for the price rather than order sequence number.
This means that the lookup function might return the wrong order if there's another order with the same price.
**Note:** this doesn't have any effect on current deployment since the lookup function isn't used for RestingOrder trees

```
C/C++

impl PartialEq for RestingOrder {
    fn eq(&self, other: &Self) -> bool {
        (self.price) == (other.price)
    }
}
```

**Recommendation:** Compare order sequence number rather than price

**Customer's response:** Not confirmed. This does nothing though as explained in response to I-06

### I-03 unnecessary update of x as child of p in rotate left/right

**Description:** At the RB tree's rotate left/right functions x is set as the child of p.
This update is unnecessary since x is already a child of p (and on the same side, right for rotate left and left for rotate right).

Rotate left:

```JavaScript
        let g_index: DataIndex = index;
        let p_index: DataIndex = self.get_right_index::<V>(g_index);
        let x_index: DataIndex = self.get_right_index::<V>(p_index);
        let y_index: DataIndex = self.get_left_index::<V>(p_index);
        let gg_index: DataIndex = self.get_parent_index::<V>(index);

        // P
        {
            // Does not use the helpers to avoid redundant NIL checks.
            let p_node: &mut RBNode<V> = get_mut_helper::<RBNode<V>>(self.data(), p_index);
            p_node.parent = gg_index;
            p_node.left = g_index;
            p_node.right = x_index;
        }
```

Rotate right:

```JavaScript
        let g_index: DataIndex = index;
        let p_index: DataIndex = self.get_left_index::<V>(g_index);
        let x_index: DataIndex = self.get_left_index::<V>(p_index);
        let y_index: DataIndex = self.get_right_index::<V>(p_index);
        let gg_index: DataIndex = self.get_parent_index::<V>(index);

        // P
        {
            // Does not use the helpers to avoid redundant NIL checks.
            let p_node: &mut RBNode<V> = get_mut_helper::<RBNode<V>>(self.data(), p_index);
            p_node.parent = gg_index;
            p_node.left = x_index;
            p_node.right = g_index;
        }
```

**Recommendation:** Remove this unnecessary update

**Customer's response:** Fixed in PR #216 .

**Fix Review:** Fix confirmed

## I-04 - Orders of zero price can be added to tree

**Description:** According to the whitepaper, Manifest is expected to enforce a minimum order size to ensure trading can occur on the curve. Resting with price zero should be prevented

> Manifest removes the tradeoff between large ticks or large minimum order quantities altogether by making minimum order size atomic and tick size subatomic.
>
> $$Price\ Tick\ Size = 10^{-18}$$ Quote Atoms / Base Atom
> $$Min\ Order\ Size = 1$$

However, on the `place_order` function, there's no check to prevent resting/posting an order with a zero price.

**Recommendation:** Add a check to prevent resting orders with zero price

**Customer's response:** Fixed in PR #122

**Fix Review:** Fix confirmed

### I-05 - `swap_nodes()` doesn't support swapping with node root as the second node

**Description:** The function `swap_nodes()` does not cover the case where the second node is the root. This should not impact the correctness of the system, since the function is called on internal nodes and their successor. Nevertheless, the function can be modified to support the second case as well and prevent unexpected behavior if ever used in future unforeseen scenarios.

**Recommendation:** Either support also swapping with the root node as the second node, or clearly document that this function is only intended to be used from `remove_by_index`, for example, by renaming it into `swap_nodes_for_remove`.

**Customer's response:** Not confirmed. Renaming to be clearer that it is an internal implementation detail only, not a general swap function. Update in PR #260.

### I-06 - Canceling an order might run out of compute units when no index hint is provided

**Description:** When no hint index is provided for the users, the `cancel_order()` function is called. This function iterates over both trees to search for the relevant sequence number to delete. This means if the trees are large, this iteration may end up needing to iterate so many times that the sequence number has not yet been found.

**Recommendation:** Consider using the lookup function to find the order (after fixing it, see I-02. That would require the user to also pass the price in order to use the lookup function)

**Customer's response:** Not confirmed. Working as intended. We want to encourage O(1) access and don't really care if people hit the worst case on naive usage. The suggestion does not work. The tree is sorted by price, so a sequence number equality check doesn't make a difference. We would need a separate data structure with cmp operators for sequence numbers.

### I-07 - Dead code in `get_next_higher_index()`

**Description:** The second part of the `get_next_higher_index()` isn't reachable under current implementation, since the function is only called from `remove_by_index()`, where the index is internal and the successor is below.

```
JavaScript
   fn get_next_higher_index<V: Payload>(&'a self, index: DataIndex) -> DataIndex {
        if index == NIL {
            return NIL;
        }
        // Successor is below us.
        if self.get_right_index::<V>(index) != NIL {
            let mut current_index: DataIndex = self.get_right_index::<V>(index);
            while self.get_left_index::<V>(current_index) != NIL {
                current_index = self.get_left_index::<V>(current_index);
            }
            return current_index;
        }

        // Successor is above, keep going up while we are the right child
        let mut current_index: DataIndex = index;
        while self.is_right_child::<V>(current_index) {
            current_index = self.get_parent_index::<V>(current_index);
        }
        current_index = self.get_parent_index::<V>(current_index);

        current_index
}
```

**Recommendation:** We have checked this function as part of the verification of remove_by_index. In this context, the additional branch is not needed and is confusing. However, if the function is intended to use elsewhere it can be kept unchanged. However, we want to stress that we did not verify the code that is dead under current use.

**Customer's response:** Fixed in PR #217 .

**Fix Review:** Fix confirmed

### I-08 - Duplicate variable declaration in `insert_fix()`

**Description:** The `parent_color` variable is declared and retrieved twice in the `insert_fix()` function. The second declaration isn't needed since the parent color didn't change since then.

JavaScript

```
        // Check the color of the parent. If it is black, then nothing left to do.
        let parent_index: DataIndex = self.get_parent_index::<V>(index_to_fix);
        let parent_color: Color = self.get_color::<V>(parent_index);

      // ........................



      let grandparent_color: Color = self.get_color::<V>(grandparent_index);
        let parent_color: Color = self.get_color::<V>(parent_index);
        let parent_is_left: bool = self.is_left_child::<V>(parent_index);
        let current_is_left: bool = self.is_left_child::<V>(index_to_fix);
```

**Recommendation:** Remove the second declaration

**Customer's response:** Fixed in PR #219 .

**Fix Review:** Fix confirmed

### I-09 - if blocks can be refactored into if-else

**Description:** The following if code blocks in the `insert_fix()` function can be refactored into if-else code blocks since they're mutually exclusive

```
C/C++
        if parent_is_left && current_is_left {
             // implementation details
        }
        let index_to_fix_color: Color = self.get_color::<V>(index_to_fix);
        // Case III: Uncle is black, left right
        if parent_is_left && !current_is_left {
             // implementation details
        }
        // Case IV: Uncle is black, right right
        if !parent_is_left && !current_is_left {
             // implementation details
        }
        // Case V: Uncle is black, right left
        if !parent_is_left && current_is_left {
             // implementation details
        }
```

**Recommendation:** Refactor into if-else blocks

**Customer's response:** Fixed in PR #218

**Fix Review:** Fix confirmed

### I-10 - Check that `in_atoms` is backed up by actual funds

**Description:** The `Swap` instruction takes an argument `in_atoms` that represents the maximal amount that can be traded during the swap. This value is unchecked and can be arbitrary large. While overall, `Swap` is safe because the traded amount is taken from the user-provided wallet, and, therefore, will always be backed up, during the execution of the instruction this is not known. We recommend checking that the `in_atoms` is not greater than the amount of tokens in the user-provided Token Account.

Currently, this does not present a problem because the code uses checked arithmetic exclusively. However, in the future, if the code is optimized to reduce the use of checked arithmetic, an unchecked parameter may prevent optimizations or cause bugs.

**Recommendation:** Check that `in_atoms` is bounded above by `amount` in the `trader_base_account` or `trader_quote_amount` as determined by the `is_base_in` parameter.

**Customer's response:** Fixed in PR #258

**Fix Review:** Fix confirmed

### I-11 - Avoid complications of Effective Price by rounding in favor of large orders

**Description:** The effective price adds several complications in the code and introduces more problems where a trader may not get the best price. Issue L-03 is another problem caused by this.

The reason for introducing this effective price is to disincentivize traders to post multiple small orders in the order book where they would profit from rounding errors. This is because swap amounts are rounded in favor of the order in the order book (the maker). There is another very easy way to solve this problem: round in favor of the larger order.

**Recommendation:** In swap instead of rounding in favor of the maker, the code should round in favor of the taker when he matches the full order and in favor of the maker when the order is only partially matched. This way, every user (taker and maker) is protected against the other party introducing micro-trades. The effective price is no longer needed and orders can be sorted by price instead. Partially fulfilled orders can keep their place in the book.

```
C/C++
// on full match: round in favor of the taker
// on partial match: round in favor of the maker
let full_match: bool =
               remaining_base_atoms >= other_order.get_num_base_atoms();
let quote_atoms_traded: QuoteAtoms = matched_price.checked_quote_for_base(
    base_atoms_traded,
    round_up: is_bid != full_match,
)?;
```

This has a nice side-effect: When posting a large order as a taker, only the very large match will be a partial match, so there will be only one instance where the rounding is against the trader. This guarantees that the maximum loss due to rounding errors is less than a single atom. This also holds for posting a large order as a maker, as in this case there is only at most one full match that is rounded against the maker. And even if you create a new order where part is matched with the order book and the remaining order is added to the order book, the matching parts will all round in the favor of the trader as they match fully and there will only be one time where the rounding is against the trader. To summarize: there will be at most one loss due to rounding for every order less than one quote atom will be lost due to this rounding.

The effective price is no longer necessary, because in the case of a full match the effective price is better than the price, so the taker is guaranteed that the price he gets is the best price even if rounding occurs.

**Customer's response:** Adopted in PR #108.

**Fix Review:** Fix confirmed

### I-12 - Check that cancel_order_by_index allows a trader to cancel only their own order.

**Description:** The function cancel_order_by_index removes the order at order_index that is taken as parameter. Function cancel_order_by_index does not check that the order at order_index belongs to the trader at trader_index (also taken as a parameter). As per the design, this check is to be made by any function that calls cancel_order_by_index. The Manifest code ensures this currently, however it is possible to make a mistake here with future changes.

**Recommendation:** Add a check in cancel_order_by_index that the order at order_index belongs to the trader at trader_index.

**Customer's response:** Not confirmed. Working as intended.

### I-13 - Remove recursion in Red-Black Tree implementation

**Description:** Functions `remove_fix`, `insert_fix`, and `lookup_index` in `red_black_tree.rs` used tail recursion. The use of recursion makes the code less predictable because it depends on the available stack space and recursion limit in SBF VM. Recursion also complicates FV because FV must account for the available stack space.

**Recommendation:** Replace recursion by loops

**Customer's response:** Fixed in PR #174

**Fix Review:** Fix confirmed

### I-14 - `swap_nodes()` doesn't support swapping nodes that are parent and child

**Description:** The function `swap_nodes()` does not cover three cases:

1. The second node is the left child of the first
2. The first node is the right child of the second
3. The first node is the left child of the second

In these cases, the pointers to the left and right children of the nodes after the execution of the function result inconsistent. This should not impact the correctness of the system, since the function is called on internal nodes and their successor. This issue is related to I-05. Observe that the case in which the second node is the right child of the first is correctly supported.

**Recommendation:** Change the specification of the function `swap_nodes` to make it explicit that it is not intended to be used in the aforementioned cases 1-3.

**Customer's response:** Fixed in PR #224 .

**Fix Review:** Fix confirmed

certora

# 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. |

## General Assumptions and Simplifications

1. We used Rust Compiler version 1.18.16 to generate SBF code.
2. Use of mocks
    a. Mock for red-black tree: we employ a mock implementation of the red-black tree to isolate and validate the correctness of the orderbook independently of the specific tree implementation details. In particular, we use an abstraction in which we can access precise information of two main traders, and any access to other traders results in an error. Similarly, at most one bid or ask order can be accessed precisely, while all the accesses to other orders results in an error.
    b. Mock for QuoteAtomsPerBaseAtom: we keep the price to be the same format as original [u64; 2] but restrict the price between [0, 0] and [0, u32::MAX]. We also consider only up to 4 decimal points.
3. We disable global orders by munging the is_global method of RestingOrders to always return false.

# Formal Verification Properties

## Verifying “no loss of funds”

- The “no loss of funds” invariant, in English, is stated as follows: the amount in the vault matches the sum of balances contained in the orders and seats. We want to show that this invariant holds for both base and quote tokens.
- To calculate the balances contained in orders, we introduce two additional fields ‘orderbook_base_atoms’ and ‘orderbook_quote_atoms’ for MarketFixed to store them. We also introduce an additional method of the RestingOrder called ‘get_orderbook_atoms’ to calculate the balance stored in an order. The fields ‘orderbook_base_atoms’ and ‘orderbook_quote_atoms’ are updated wherever relevant.
- Similarly, we introduce additional fields `withdrawable_base_atoms` and `withdrawable_quote_atoms` to store the sum of seat balances. These fields are also updated wherever relevant.
- With above changes, we can now state the “no loss of funds” as follows:

  vault_base == orderbook_base + withdrawable_base

  vault_quote == orderbook_quote + withdrawable_quote
- We show that “no loss of funds” is an invariant in the following way: for each function, we assume that “no loss of funds” holds before the function, we execute the function, and finally assert that “no loss of funds” holds at the end. All rules for “no loss of funds” follow this pattern.
- We show “no loss of funds” for deposit, withdraw, rest_remaining, cancel_order_by_index, cancel_order, place_order and swap. Some functions, like claim_seat and release_seat, do not change any quantity related to “no loss of funds”, so we do not consider them. For batch_update, we check that it calls cancel_order and place_order at the end. These rules for batch_update are covered in the next section Rules for Integrity.
- We use checked arithmetic when assuming the invariant, and saturating arithmetic when asserting the invariant. This allows catching the overflow/underflow in computation.
- Due to L-01, we changed the direction of rounding in cancel_order_by_index. We also changed the code in place_order after the orders are matched: originally, the maker is increased according to the matched order, followed by the taker being decreased. This may cause an overflow when updating the withdrawable balances. To avoid this, we change the order of increase and decrease. We first decrease the taker, followed by increasing the maker.

### Module General Assumptions

- Munging in place_order: we extract the matching loop in place_order into a separate function called place_single_order. We verify “no loss of funds” for place_single_order.
- Munging in cancel_order: we unravel the bid/ask loop in cancel_order manually.
- Mock for red-black tree: as described in Sec. General Assumptions.
- Mocks for price: as described in Sec. General Assumptions.
- All rules for “no loss of funds” make the following assumption on the state of the market: (i) the Pubkeys of the market_base_vault and market quote vault are unequal; (ii) there are two traders and one order in the

market; (iii) the Pubkeys of the two traders are unequal; (iv) the order on the market is not global and has nondeterministic price and base atoms.

## Module Properties

### **P-01.** "no loss of funds" - deposit

Status: Verified

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_deposit_base** | Verified | *This rule checks "no loss of funds" when base atoms are deposited. Additional assumption: the Pubkeys of trader's vault and market's vault are unequal.* | *Report* |
| **rule_deposit_quote** | Verified | *This rule checks "no loss of funds" when quote atoms are deposited. Additional assumption: the Pubkeys of trader's vault and market's vault are unequal.* | *Report* |

### **P-02.** "no loss of funds" - withdraw

Status: Verified after fix

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_withdraw_base** | Verified after fix | *This rule checks "no loss of funds" when base atoms are withdrawn. Additional assumption: the Pubkeys of trader's vault and market's vault are unequal. The violation is due to H-01. The rule passes after applying the fix.* | *Report*<br>*Report after fix* |

| | | | |
|---|---|---|---|
| **rule_withdraw_quote** | Verified after fix | *This rule checks "no loss of funds" when quote atoms are withdrawn. Additional assumption: the Pubkeys of trader's vault and market's vault are unequal. The violation is due to H-01. The rule passes after applying the fix.* | *Report*<br>*Report after fix* |

## P-03. "no loss of funds" - rest_remaining

Status: Verified

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_rest_remaining_bid** | Verified | *This rule checks "no loss of funds" when rest_remaining inserts a bid order into the market.* | *Report* |
| **rule_rest_remaining_ask** | Verified | *This rule checks "no loss of funds" when rest_remaining inserts an ask order into the market.* | *Report* |

## P-04. "no loss of funds" - cancel_order_by_index

Status: Verified after fix

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_cancel_order_by_index_bid** | Verified after fix | *This rule checks "no loss of funds" when cancel_order_by_index cancels a bid order. The violation is due to L-01. Fixing L-01 makes the rule pass.* | *Report*<br>*Report after fix* |

| | | | |
|---|---|---|---|
| **rule_cancel_order_by_index_ask** | Verified | *This rule checks "no loss of funds" when cancel_order_by_index cancels an ask order.* | *Report* |

## P-05. "no loss of funds" - cancel_order

Status: Verified after fix

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_cancel_order_bid** | Verified after fix | *This rule checks "no loss of funds" when cancel_order cancels a bid order. The violation is due to L-01. Fixing L-01 makes the rule pass.* | *Report*<br>*Report after fix* |
| **rule_cancel_order_ask** | Verified | *This rule checks "no loss of funds" when cancel_order cancels an ask order.* | *Report* |

## P-06. "no loss of funds" - place_order

Status:

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_place_single_order_canceled_bid** | Verified | *This rule checks "no loss of funds" when a bid order does not match with an existing ask order because it is expired.* | *Report* |

| | | | |
|---|---|---|---|
| **rule_place_single_order_canceled_ask** | Verified | *This rule checks "no loss of funds" when an ask order does not match with an existing bid order because it is expired.* | *Report* |
| **rule_place_single_order_unmatched_bid** | Verified | *This rule checks "no loss of funds" when a bid order does not match with an existing ask order because of its price.* | *Report* |
| **rule_place_single_order_unmatched_ask** | Verified | *This rule checks "no loss of funds" when an ask order does not match with an existing bid order because of its price.* | *Report* |
| **rule_place_single_order_full_match_bid** | Verified | *This rule checks "no loss of funds" when a bid order fully matches with an existing ask order.* | *Report* |
| **rule_place_single_order_full_match_ask** | Verified | *This rule checks "no loss of funds" when an ask order fully matches with an existing bid order.* | *Report* |
| **rule_place_single_order_partial_match_bid** | Verified | *This rule checks "no loss of funds" when a bid order partially matches with an existing ask order.* | *Report* |
| **rule_place_single_order_partial_match_ask** | Verified | *This rule checks "no loss of funds" when an ask order partially matches with an existing bid order.* | *Report* |

## P-07. “no loss of funds” - swap

Status: Verified

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_swap_base_exact** | Verified | *This rule checks “no loss of funds” for swap with is_base_in = true, is_exact_in = true* | *Report* |
| **rule_swap_base_not_exact** | Verified | *This rule checks “no loss of funds” for swap with is_base_in = true, is_exact_in = false* | *Report* |
| **rule_swap_quote_exact** | Verified | *This rule checks “no loss of funds” for swap with is_base_in = false, is_exact_in = true* | *Report* |
| **rule_swap_quote_not_exact** | Verified | *This rule checks “no loss of funds” for swap with is_base_in = false, is_exact_in = false* | *Report* |

# Formal Verification Properties

## Rules for the integrity of deposit, withdrawal, order cancelation and batch update

The rules in this module check the correctness and integrity of deposit, withdraw, cancel_order_by_index and batch_update.

### Module General Assumptions

- Mock for a red-black tree: as described in Sec. General Assumptions.
- Mocks for price: as described in Sec. General Assumptions.

### Module Properties

**P-01.** Integrity of deposit

| Status: Verified | |
|---|---|

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_deposit_deposits** | Verified | *This rule checks for deposit the following two properties: (i) the trader balances and vault amounts are updated correctly and (ii) no other trader's balance is affected.* | *Report* |

## P-02. Integrity of withdraw

Status: Verified after fix

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_withdraw_withdraws** | Verified after fix | *This rule checks for withdraw the following two properties: (i) the trader balances and vault amounts are updated correctly and (ii) no other trader's balance is affected. The violation is due to H-01. The rule passes after applying the fix.* | *Report*<br>*Report after fix* |

## P-03. No unexpected revert on cancel_order_by_index

Status: Verified

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_cancel_order_by_index_no_revert_bid** | Verified | *This rule verifies that cancel_order_by_index does not have an unexpected revert, i.e., not caused by require!(..). Same assumption on the pre-condition of the market as "no loss of funds". An additional assumption that seat balance does not overflow. The order is assumed to be a bid order.* | *Report* |
| **rule_cancel_order_by_index_no_revert_ask** | Verified | *Same as above, except the order is assumed to be an ask order.* | *Report* |

## P-04. Integrity of batch_update

Status: Verified

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_integrity_of_batch_update_cancel_bid** | Verified | *This rule checks that batch_update, when called on 1 bid order to cancel and no orders to place, invokes cancel_order correctly. This establishes no loss of funds for this case.* | Report |
| **rule_integrity_of_batch_update_cancel_ask** | Verified | *This rule checks that batch_update, when called on 1 ask order to cancel and no orders to place, invokes cancel_order correctly. This establishes no loss of funds for this case.* | Report |
| **rule_integrity_of_batch_update_cancel_hint_bid** | Verified | *This rule checks that batch_update, when called on 1 bid order to cancel with hint and no orders to place, invokes cancel_order_by_index correctly. This establishes no loss of funds for this case.* | Report |
| **rule_integrity_of_batch_update_cancel_hint_ask** | Verified | *This rule checks that batch_update, when called on 1 ask order to cancel with hint and no orders to place, invokes cancel_order_by_index correctly. This establishes no loss of funds for this case.* | Report |
| **rule_integrity_of_batch_update_place_order_bid** | Verified | *This rule checks that batch_update, when called on 1 bid order to place and no orders to cancel, invokes place_order correctly. This establishes no loss of funds for this case.* | Report |
| **rule_integrity_of_batch_update_place_order_ask** | Verified | *This rule checks that batch_update, when called on 1 ask order to place and no orders to cancel, invokes place_order correctly. This establishes no loss of funds for this case.* | Report |

# Formal Verification Properties

## Rules for the matching mechanism

The rules in this module check the correctness and integrity of the matching mechanism.

### Module General Assumptions

- Mock for a red-black tree: as described in Sec. General Assumptions.
- Mocks for price: as described in Sec. General Assumptions.
- All rules for "no loss of funds" make the following assumption on the state of the market: (i) the Pubkeys of the market_base_vault and market quote vault are unequal; (ii) there are two traders and one order in the market; (iii) the Pubkeys of the two traders are unequal; (iv) the order on the market is not global and has nondeterministic price and base atoms.

### Module Properties

**P-01.** Correctness of the matching mechanism

Status: Verified

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_matching_if_maker_order_exists_bid** | Verified | *Rules to check if a matching maker order exists in the book, then matching will happen. The order to place is a bid order.* | *Report* |
| **rule_matching_if_maker_order_exists_ask** | Verified | *Rules to check if a matching maker order exists in the book, then matching will happen. The order to place is an ask order.* | *Report* |
| **rule_crossed_prices_if_matched_bid** | Verified | *Rules to check if matching happened, then prices must have crossed. The order to place is a bid order.* | *Report* |
| **rule_crossed_prices_if_matched_ask** | Verified | *Rules to check if matching happened, then prices must have crossed. The order to place is an ask order.* | *Report* |

| | | | |
|---|---|---|---|
| **rule_place_single_order_full_match_balances_bid** | Verified | *Rules to verify that trader balances are modified as expected in the fully_matched case of place_single_order. The order to place is a bid order.* | Report |
| **rule_place_single_order_full_match_balances_ask** | Verified | *Rules to verify that trader balances are modified as expected in the fully_matched case of place_single_order. The order to place is an ask order.* | Report |
| **rule_place_single_order_partial_match_balances_bid** | Verified | *Rules to verify that trader balances are modified as expected in the partially_matched case of place_single_order. The order to place is a bid order.* | Report |
| **rule_place_single_order_partial_match_balances_ask** | Verified | *Rules to verify that trader balances are modified as expected in the partially_matched case of place_single_order. The order to place is an ask order.* | Report |
| **rule_matching_order_removed_if_fully_matched_bid** | Verified | *Rules to check if the maker_order is fully_matched, then it is removed. The order to place is a bid order.* | Report |
| **rule_matching_order_removed_if_fully_matched_ask** | Verified | *Rules to check if the maker_order is fully_matched, then it is removed. The order to place is an ask order.* | Report |
| **rule_matching_fully_matched_if_order_removed_bid** | Verified | *Rules to check if the maker_order was (i) not expired (ii) matched on price and (iii) removed from the tree, then it must have been fully_matched. The order to place is a bid order.* | Report |
| **rule_matching_fully_matched_if_order_removed_ask** | Verified | *Rules to check if the maker_order was (i) not expired (ii) matched on price and (iii) removed from the tree, then it must have been fully_matched. The order to place is an ask order.* | Report |
| **rule_matching_decrease_maker_order_atoms_bid** | Verified | *Rules to check that the atoms contained in the maker_order cannot increase after matching. The order to place is a bid order.* | Report |

| | | | |
|---|---|---|---|
| **rule_matching_decrease_maker_order_atoms_ask** | Verified | *Rules to check that the atoms contained in the maker_order cannot increase after matching. The order to place is an ask order.* | *Report* |

# Formal Verification Properties

## Red-Black Tree

### Module General Assumptions

- The following properties are proved for the lib/src/red_black_tree.rs module.
- The red-black tree is parametrized by the Payload type, and in our rules we use a TestOrder payload type with a single u64 field called order_id
- We verify that the functions insert_fix and remove_fix match the behavior of a reference implementation. As reference we use the 4th edition of "Introduction to Algorithms", ISBN 026204630X. The functions are described in Chapter 13, which starts at page 331. The function insert_fix corresponds to algorithm RB-Insert-Fixup described at page 339, while remove_fix corresponds to RB-Delete-Fixup described at page 351.
- The rules to check that insert_fix matches the reference implementation assume that the grandparent of the node that has to be fixed is the child of the root. Furthermore, we ensure that the nodes in the subtrees that are not directly modified by the function have nondeterministic children and have nondeterministic color.
- The rules to check that remove_fix matches the reference implementation assume that the parent of the node that has to be fixed is the child of the root. Furthermore, we ensure that the nodes in the subtrees that are not directly modified by the function have nondeterministic children and have nondeterministic color.
- As mentioned above, in addition to PR #108, we also cherry picked commit 714be51, which removes the tail recursion from the insert_fix and remove_fix functions. This makes it easier verifying the correctness of those critical parts of the code.

### Module Properties

**P-01.** Correctness of rotation

Status: Verified

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_rotate_left** | Verified | *This rule verifies that rotate_left properly sets the indices of all the nodes involved in the rotation. We verify this for a six-node tree, where the node that is* | *Report* |

| | | | |
|---|---|---|---|
| | | *being rotated has a parent, a left child, a right child, and the right child has both a left and a right child.* | |
| **rule_rotate_right** | Verified | *This rule verifies that rotate_right properly sets the indices of all the nodes involved in the rotation. We verify this for a six-node tree, where the node that is being rotated has a parent, a right child, a left child, and the left child has both a left and a right child.* | *Report* |

## P-02. `insert_fix` matches reference implementation

Status: Verified

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_insert_fix_matches_reference_no_parent** | Verified | *This rule verifies that insert_fix matches the reference implementation in case that the node has no parent, namely it is the root.* | *Report* |
| **rule_insert_fix_matches_reference_case1_left_child** | Verified | *This rule verifies that insert_fix matches the reference implementation in case 1, when the parent of the node that has to be fixed is a left child.* | *Report* |
| **rule_insert_fix_matches_reference_case2_left_child** | Verified | *This rule verifies that insert_fix matches the reference implementation in case 2, when the parent of the node that has to be fixed is a left child.* | *Report* |
| **rule_insert_fix_matches_reference_case3_left_child** | Verified | *This rule verifies that insert_fix matches the reference implementation in case 3, when the parent of the node that has to be fixed is a left child.* | *Report* |

| | | | |
|---|---|---|---|
| **rule_insert_fix_matches_reference_case1_right_child** | Verified | *This rule verifies that insert_fix matches the reference implementation in case 1, when the parent of the node that has to be fixed is a right child.* | *Report* |
| **rule_insert_fix_matches_reference_case2_right_child** | Verified | *This rule verifies that insert_fix matches the reference implementation in case 2, when the parent of the node that has to be fixed is a right child.* | *Report* |
| **rule_insert_fix_matches_reference_case3_right_child** | Verified | *This rule verifies that insert_fix matches the reference implementation in case 3, when the parent of the node that has to be fixed is a right child.* | *Report* |

### P-03. `remove_fix` matches reference implementation

Status: Verified after fix

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_remove_fix_matches_reference_case1_left_child** | Verified | *This rule verifies that remove_fix matches the reference implementation in case 1, when the node that has to be fixed is a left child. In the reference implementation they recolor the sibling and the parent, and then they perform a rotation. After that, they keep considering the other cases, since case 1 can be then reduced to one of the subsequent cases. This is not necessary in remove_fix, since the index that is returned is index_1, which will then be handled in the while loop in the function remove in a subsequent iteration. Therefore, we check the state of case 1 after executing the lines 5-8 in the pseudocode.* | *Report* |

| | | | |
|---|---|---|---|
| **rule_remove_fix_matches_reference_case2_left_child** | Verified | *This rule verifies that remove_fix matches the reference implementation in case 2, when the node that has to be fixed is a left child.* | *Report* |
| **rule_remove_fix_matches_reference_case3_left_child** | Verified after fix | *This rule verifies that remove_fix matches the reference implementation in case 3, when the node that has to be fixed is a left child. This is due to L-07.* | *Report before fix*<br>*Report after fix* |
| **rule_remove_fix_matches_reference_case4_left_child** | Verified | *This rule verifies that remove_fix matches the reference implementation in case 4, when the node that has to be fixed is a left child.* | *Report* |
| **rule_remove_fix_matches_reference_case1_right_child** | Verified | *This rule verifies that remove_fix matches the reference implementation in case 1, when the node that has to be fixed is a right child. In the reference implementation they recolor the sibling and the parent, and then they perform a rotation. After that, they keep considering the other cases, since case 1 can be then reduced to one of the subsequent cases. This is not necessary in remove_fix, since the index that is returned is index_1, which will then be handled in the while loop in the function remove in a subsequent iteration. Therefore, we check the state of case 1 after executing the lines 26-29 in the pseudocode.* | *Report* |
| **rule_remove_fix_matches_reference_case2_right_child** | Verified | *This rule verifies that remove_fix matches the reference implementation in case 2, when the node that has to be fixed is a right child.* | *Report* |
| **rule_remove_fix_matches_reference_case3_right_child** | Verified after fix | *This rule verifies that remove_fix matches the reference implementation in case 3, when the node that has to be fixed is a right child. This is due to L-07.* | *Report before fix*<br>*Report after fix* |
| **rule_remove_fix_matches_reference_case4_right_child** | Verified | *This rule verifies that remove_fix matches the reference implementation in case 4, when the node that has to be fixed is a right child.* | *Report* |

## P-04. `insert` correctly updates `max_index`

Status: Verified

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_insert_updates_max_index_empty_tree** | Verified | *This rule verifies that max_index is correctly updated after inserting a node in an empty tree.* | *Report* |
| **rule_insert_updates_max_index_non_empty_tree_max** | Verified | *This rule verifies that max_index is correctly updated after inserting an element that is greater than any other in a non-empty tree. We verify this for a two-node tree with nondeterministic values by inserting a new nondeterministic value that is greater than any other. The left children of the two nodes are nondeterministic and represent arbitrary subtrees.* | *Report* |
| **rule_insert_updates_max_index_non_empty_tree_not_max** | Verified | *This rule verifies that max_index is not updated after inserting an element that is smaller than the maximum in a non-empty tree. We verify this for a two-node tree with nondeterministic values by inserting a new nondeterministic value that is smaller than the max element. The left child of the root is nondeterministic and represents an arbitrary subtree.* | *Report* |

## P-05. `remove` correctly updates `max_index`

Status: Verified

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_remove_updates_max_index_single_node_tree** | Verified | *This rule verifies that max_index is correctly updated after removing a node from a single-node tree.* | *Report* |
| **rule_remove_updates_max_index_non_empty_tree_max** | Verified | *This rule verifies that max_index is correctly updated after removing the max element in a non-empty tree. We verify this for a three-node tree with nondeterministic values by removing the max element. The left children of the two nodes that will not be removed are nondeterministic and represent arbitrary subtrees.* | *Report* |
| **rule_remove_updates_max_index_non_empty_tree_not_max** | Verified | *This rule verifies that max_index is not updated after removing an element that is not the max in a non-empty tree. We verify this for a three-node tree with nondeterministic values by removing the right child of the root. The left child of the root is nondeterministic and represents an arbitrary subtree.* | *Report* |

## P-06. Correctness of swap_nodes

Status: Verified after Fix

| Rule Name | Status | Description | Link to rule report |
|---|---|---|---|
| **rule_swap_internal_nodes_left_children** | Verified | *This rule verifies that swap_nodes behaves as expected in the case that the two nodes are internal nodes with parents and children. Both nodes are left children.* | *Report* |
| **rule_swap_internal_nodes_right_children** | Verified | *This rule verifies that swap_nodes behaves as expected in the case that the two nodes are internal nodes with parents and children. Both nodes are right children.* | *Report* |
| **rule_swap_internal_nodes_first_is_root** | Verified | *This rule verifies that swap_nodes behaves as expected in the case that the first node is the root and the second is an internal node that is not adjacent to the first.* | *Report* |
| **rule_swap_internal_nodes_second_is_root** | Verified after fix | *This rule verifies that swap_nodes behaves as expected in the case that the second node is the root and the first is an internal node that is not adjacent to the second. This is due to I-05.*<br>*After the fix, the rule is no longer relevant since the spec changed.* | *Report* |
| **rule_swap_nodes_with_one_child_left_right** | Verified | *This rule verifies that swap_nodes behaves as expected in the case that the two nodes have only one child, respectively the left and the right.* | *Report* |
| **rule_swap_nodes_with_one_child_right_left** | Verified | *This rule verifies that swap_nodes behaves as expected in the case that the two nodes have only one child, respectively the right and the left.* | *Report* |

| | | | |
|---|---|---|---|
| **rule_swap_leaves** | Verified | *This rule verifies that swap_nodes behaves as expected in the case that the two nodes are leaves.* | *Report* |
| **rule_swap_parent_right_child** | Verified | *This rule verified that swap_nodes behaves as expected in the case that the second node is the right child of the first.* | *Report* |
| **rule_swap_parent_left_child** | Verified after fix | *This rule verified that swap_nodes behaves as expected in the case that the second node is the left child of the first. This is due to I-14.* | *Report before fix*<br>*Report after fix* |
| **rule_swap_right_child_parent** | Verified after fix | *This rule verified that swap_nodes behaves as expected in the case that the first node is the right child of the second. This is due to I-14.* | *Report before fix*<br>*Report after fix* |
| **rule_swap_left_child_parent** | Verified after fix | *This rule verified that swap_nodes behaves as expected in the case that the first node is the left child of the second. This is due to I-14.* | *Report before fix*<br>*Report after fix* |

# Disclaimer

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.

certora

# Appendix - FV Plan for Red-Black Tree

**Context**: We want to show that the implementation of the red-black-tree in lib/src/red_black_tree.rs is a correct implementation of a Red-Black-Tree data structure.

## General Approach

We use the Certora Prover to show that the code faithfully implements an algorithm of a reference implementation. A reference implementation is either

1. a known to be correct algorithm from a text-book on algorithms;
2. a modification created by Certora of an algorithm from (a) that fits closer to the given implementation; or
3. An algorithm that has been previously verified with a manual theorem prover (e.g., Coq, Lean, or Dafny)

Note that a reference implementation is not deterministic. Strictly speaking, it is not an algorithm. It is a specification of all individual functions (e.g., insert, insert_fix, rotate_left, etc.) such that any implementation that satisfies the specification individually is a correct implementation of the Red-Black-Tree. That is, the tree is a sorted balanced binary tree.

This is a typical approach for code verification in which code is verified against low-level specifications (i.e., a reference implementation) and the proof of correctness of reference implementation is done independently of the code. Since the proof of reference implementation does not change between different implementations, it is done once. Most times, informally in a textbook or paper that presents the algorithm.

The above is required because verification of an abstract data structure is unbounded. The proof is parametric on data structure size. A proof on code, on the other hand, takes into account the restriction placed by finitely available memory and finite representation of numbers.

## Specific Approach

We have found several formally verified reference implementations. Unfortunately, they do not follow the algorithm implemented by manifest. Specifically, it seems that the left-leaning-red-black-tree variant is easier to implement, and is, therefore, often used in verification case studies [1].

We have found that the algorithm in "Introduction to Algorithms 4th edition - Cormen, Leiserson, Rivest, Stein" (Chapter 13) is very close to the current implementation. Therefore, we have adopted it as the reference implementation. If we are unable to show that the implementation refines the reference implementation, we will either adjust the reference implementation (if the proof of correctness of the modification is obvious), or suggest how to modify the implementation to match the reference.

## Detailed Verification Properties

The following lists the properties that we are establishing of the functions in red_black_tree.rs

1. fn lookup_index()
    a. This function is recursive. We cannot guarantee that it will not run out of stack.
2. fn insert
    a. Correctly updates root_index and max_index
    b. Correctly calls BST insertion (insert_node_no_fix)
    c. Always calls insert_fix to fix any issues introduced by insertion
3. fn insert_fix(index)
    a. The function correctly restores the local red-black-tree properties by correctly updating the tree region determined by index: index (C), parent (P), grandparent (G), uncle (U), and grandgrandparent (GG) as determined by the reference implementation.
    b. Note that the color of C is Red on entry to this function.
    c. We will identify if the function is correct
4. fn remove_by_index(index)
    a. Correctly updates root_index and max_index
    b. If index has both children, it is swapped with the correct node lower in the tree
    c. Correctly handles easy cases of deletion
    d. Calls remove_fix on index only under the conditions that are required by the reference implementation
5. fn remove_fix(child_index, parent_index)

a. This function is similar to insert_fix. If the node to be fixed is black, there are four main cases, depending on: 1) the color of the sibling 2) the color of the children of the sibling. These cases are: 1) the sibling is red 2) the sibling is black and both children are black 3) the sibling is black, the left child is red, and the right child is black 4) the sibling is black, the left child is black, and the right child is red. Each case has a specular case depending on whether the node the function is fixing is a left child or right child.

When verifying each function, we will include the code of all functions it calls, unless the callee is to be verified separately. For example, verification of insert_fix will include the code of rotate_left, but verification of insert uses the verified summary of insert_fix.

We identified insert_fix, remove_fix, swap_nodes and core steps in remove_by_index as most complex. The code is difficult to understand. Has many branches. These functions are known to be the hardest part of a Red-Black-Tree implementation. Therefore, we started our verification effort on these functions. Any other function that might not be formally verified has been manually compared to the reference implementation.