Formal Verification of Orchid’s Smart Contracts

December 19, 2019

Summary

This report describes findings from a formal verification project undertaken by Certora to study Orchid’s smart contracts. Certora analyzed four Ethereum contracts written in Solidity by Orchid: the lottery, the token, the directory, and the curator.

The latest git commit analyzed was `c1d839c1b8c28eed361d5a2c144f421d09ba7e61`.

Disclaimer

Certora Prover takes as input both a smart contract as well as a specification of the contract’s behavior. The prover then attempts to formally verify that the contract satisfies the specification in all scenarios. Importantly, the Certora Prover can make guarantees **only subject to the provided specification**. Any cases not described by the specification are not checked by Certora Prover. Furthermore, because loop invariants were not provided, loops were analyzed using bounded model checking. Lastly, any off-chain code is outside the scope of this analysis.

We hope that the information in this report is useful, but we provide no warranty of any kind, express or implied. The contents of this report should not be construed as a complete guarantee that the Orchid system is secure in all dimensions. In no event shall Certora or any of its employees be held 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.

Properties Verified

Properties are described in high-level English, together with the names of the corresponding formal rules that were checked by the Certora Prover. See the Appendix for the definitions of the formal rules.

Lottery

- The push() and move() public functions revert exactly when a known set of conditions hold. [push_revert_characterization, move_revert_characterization]
    - See the Appendix for details. Roughly speaking, these contracts revert if and only if balances are insufficient or if overflow would occur in the underlying token.
- The push() and move() public functions conserve value across the underlying ERC20 token and the pot balances. [push_balances, move_balances]

- For push(), the amount added to the pot should equal the amount transferred to the lottery's address in the underlying ERC20 token.
- For move(), the total value in the funder's pot (i.e., the amount plus the escrow) should not change across the call.

- Only winning tickets can be redeemed. [implied by grab_revert_characterization]
- Workflow test: users can complete a simple cycle of push(), grab(), warn(), pull(). Furthermore, whenever a user's balance is nonzero, warn will succeed. [workflow_test]

Token

- Total supply is initialized to a value that fits in a 128-bit unsigned integer. [total_supply_initial]
- Total supply never changes after initialization. [total_supply_initial]
- Token conforms to Certora's standard specification of ERC20 tokens.

Directory

- Stake amounts are zero if and only if all fields are zero [existing_stake_nonzero]
- If take() succeeds, then it has been at least delay since pull() was invoked. [pull_take_delay]
- push() inserts a new stake struct or updates existing stake struct by adding amount. [push_introduces_new_entry, push_increases_amount]
- Only pull() causes a stake amount to become zero. [only_pull_removes, stop_does_not_remove]
- Calling pull() with the full stake amount causes the stake to be removed. [removed_does_not_exist]
- All interactions with underlying token are reflected by a corresponding change in stake amount. [balance_mirrors_stakees_stop, balance_mirrors_stakees_take, balance_mirrors_stakees_pull, balance_mirrors_stakees_push, balance_mirrors_stakees]
- pull() maintains a well formed tree structure. [wellformed_tree_bounded]
    - This property is challenging to express in full generality with current constraint solving technology. We instead checked the tree well-formedness invariant for a tree with three nodes. As the Certora Prover continues to improve, we will be able to use quantifiers to check the fully general version of this rule.

Curator

- Only the owner can make changes to the set of good addresses. [curated_unchanged]
- The owner never changes. [owner_unchanged]

Findings and Discovered Issues

We encoded the above properties into formal specifications and used the Certora tool to check them.

- The lottery contract should never be called with msg.sender as its own address.
    - If msg.sender is set to the address of the lottery in a call to push() or move(), then these functions may not correctly conserve the value of the underlying ERC20 and the pot

balances, causing pot value to be minted without a corresponding transfer of ERC20 balances.
    - Because the lottery contract makes no external calls to itself, msg.sender will never be its own address. Any future versions of the lottery contract should be checked that there is no possibility of an self-directed external call.
- To avoid arithmetic overflow in the lottery and directory, token and pot balances should remain much less than 2^{128}.
    - Pot amount and escrow are stored as 128-bit integers and are manipulated using the Solidity primitive arithmetic operations, which have wraparound semantics on overflow. Should overflow occur, pot value would be irrevocably burned.
    - Because the underlying token's initial balance is $10^9 * 10^{18} = 10^{27} \approx 2^{90} \ll 2^{128}$, it is not possible to transfer enough value to the lottery to cause overflow in a single call to push(). Furthermore, it is an invariant of the lottery that all pot value is backed by value in the lottery's balance in the underlying ERC20 token. Pot values themselves therefore remain much less than 2^{128}, and are immune from overflow.
- In an earlier version of the directory contract, pull() did not correctly maintain the tree structure.
    - Specifically, a typo in a branch condition meant that when the leaf was a direct child of the node to be removed, execution always treated the leaf as if it was the right child of its parent, even if it was actually the left child. This could cause the other subtree to be incorrectly dropped from the tree, and could cause a stake to have a left child pointer equal to its right child pointer.
    - This issue was addressed during the course of our review, and the version of the directory contract from the git commit mentioned above has been checked to preserve the tree structure.

Appendix

This appendix contains the formalized specifications for Orchid's contracts.

lottery.cvl

```
rule look_revert_characterization(address funder, address signer) {
        env e;
        invoke harness_look(e, funder, signer);
        assert lastReverted <=> e.msg.value != 0;
}

rule push_revert_characterization(address signer, uint128 total, uint128 escrow) {
        uint max_uint128 = 340282366920938463463374607431768211455;
        uint max_uint256 =
115792089237316195423570985008687907853269984665640564039457584007913129639935;

        require
          0 <= escrow && escrow <= max_uint128 &&
          0 <= total  && total  <= max_uint128;

        env e_pre; env e_f;
        address funder = e_pre.msg.sender;
        require e_f.msg.sender == funder;

        uint256 pre_bal_funder = sinvoke harness_token_balanceOf(e_pre, funder);
        uint256 pre_bal_this = sinvoke harness_token_balanceOf(e_pre, currentContract);
        uint256 pre_allowance_funder = sinvoke harness_token_allowance(e_pre, funder,
currentContract);

    invoke push(e_f, signer, total, escrow);

        bool push_reverted = lastReverted;

        bool push_no_value = e_f.msg.value == 0;
        bool total_ge_escrow = total >= escrow;
        bool funder_nonzero = funder != 0;
        bool total_le_bal_funder = total <= pre_bal_funder;
        bool no_add_overflow = pre_bal_this + total <= max_uint256 || funder ==
currentContract;  // potential issue: aliasing
        bool sufficient_allowance = total <= pre_allowance_funder;

        assert push_reverted <=> (
                !push_no_value ||
                !total_ge_escrow ||
                !funder_nonzero ||
                !total_le_bal_funder ||
                !no_add_overflow ||
                !sufficient_allowance
        );
}

rule push_balances(address signer, uint128 total, uint128 escrow) {
        uint max_uint128 = 340282366920938463463374607431768211455;
        uint max_uint256 =
115792089237316195423570985008687907853269984665640564039457584007913129639935;
        uint total_supply = 1000000000000000000000000000;

        require
          0 <= escrow && escrow <= max_uint128 &&
          0 <= total  && total  <= max_uint128;

        env e_pre; env e_f; env e_post;
        address funder = e_f.msg.sender;

        uint128 pre_amount;
        uint128 pre_escrow;
        pre_amount, pre_escrow, _, _, _ = sinvoke harness_look(e_pre, funder, signer);

        uint256 pre_bal_funder = sinvoke harness_token_balanceOf(e_pre, funder);
        uint256 pre_bal_this = sinvoke harness_token_balanceOf(e_pre, currentContract);
        require pre_bal_funder + pre_bal_this <= total_supply;

     sinvoke push(e_f, signer, total, escrow);

        uint128 post_amount;
        uint128 post_escrow;
        post_amount, post_escrow, _, _, _ = sinvoke harness_look(e_post, funder,
signer);

        uint256 post_bal_funder = sinvoke harness_token_balanceOf(e_post, funder);
        uint256 post_bal_this = sinvoke harness_token_balanceOf(e_post,
currentContract);

        uint128 amount = total - escrow;
```

```
    require pre_amount + amount <= max_uint128;  // potential issue: overflow
    require pre_escrow + escrow <= max_uint128;  // potential issue: overflow

      address this = currentContract;
      require this != funder;  // potential issue: change in ERC20 balance may not
correspond to change in pot

      assert
        pre_bal_funder + pre_amount + pre_escrow ==
        post_bal_funder + post_amount + post_escrow;
}

rule move_revert_characterization(address signer, uint128 amount) {
      uint max_uint128 = 340282366920938463463374607431768211455;

      require 0 <= amount && amount <= max_uint128;

      env e_pre; env e_f;
      address funder = e_pre.msg.sender;
      require e_f.msg.sender == funder;

      uint128 pre_amount;
      uint128 pre_escrow;
      pre_amount, pre_escrow, _, _, _ = sinvoke harness_look(e_pre, funder, signer);

      require e_f.msg.value == 0;
      invoke move(e_f, signer, amount);
      bool move_reverted = lastReverted;

      bool pot_amount_sufficient = pre_amount >= amount;

      assert move_reverted <=> !pot_amount_sufficient;
}

rule move_balances(address signer, uint128 amount) {
      uint max_uint128 = 340282366920938463463374607431768211455;

      require 0 <= amount && amount <= max_uint128;

      env e_pre; env e_f; env e_post;
      address funder = e_pre.msg.sender;
```

```
        uint128 pre_amount;
        uint128 pre_escrow;
        pre_amount, pre_escrow, _, _, _ = sinvoke harness_look(e_pre, funder, signer);

        require pre_escrow + amount <= max_uint128;  // potential issue: overflow

        sinvoke move(e_f, signer, amount);

        uint128 post_amount;
        uint128 post_escrow;
        post_amount, post_escrow, _, _, _ = sinvoke harness_look(e_post, funder,
signer);

        assert pre_amount + pre_escrow == post_amount + post_escrow;
}

rule harness_keccak_abi_bytes32_revert_cfharacterization(bytes32 x) {
        env e;
        invoke harness_keccak_abi_bytes32(e, x);
        bool reverted = lastReverted;
        assert reverted <=> e.msg.value != 0;
}

rule grab_revert_characterization() {
        env e_pre; env e_f;

    bytes32 reveal; bytes32 commit;
    uint256 issued; bytes32 nonce;
    uint8 v; bytes32 r; bytes32 s;
    uint128 amount; uint128 ratio;
    uint256 start; uint128 range;
    address funder; address recipient;

        bytes32 hash_secret = sinvoke harness_keccak_abi_bytes32(e_pre, reveal);
        uint128 hash_secret_nonce = sinvoke
harness_keccak_abi_bytes32_bytes32_as_uint128(e_pre, reveal, nonce);

        invoke grab_harness(e_f, reveal, commit, issued, nonce, v, r, s, amount, ratio,
start, range, funder, recipient);
        bool grab_reverted = lastReverted;
        bool hash_of_secret_correct = hash_secret == commit;
        bool hash_of_secret_and_nonce_winning = hash_secret_nonce <= ratio;
        assert (
                !hash_of_secret_correct ||
                !hash_of_secret_and_nonce_winning
        ) => grab_reverted;
}

rule workflow_test() {
        uint max_uint128 = 340282366920938463463374607431768211455;
        uint max_balance = 1000000000000000000000000000;

        env e_push;
        address funder = e_push.msg.sender;

        env e_pre;
        uint256 funder_balance_pre = sinvoke harness_token_balanceOf(e_pre, funder);
        uint256 lottery_balance_pre = sinvoke harness_token_balanceOf(e_pre,
currentContract);
        require funder_balance_pre + lottery_balance_pre <= max_balance;

        address signer;
        uint128 amount; require 0 <= amount && amount <= max_uint128;
        uint128 total; require 0 <= total && total <= max_uint128;

        sinvoke push(e_push, signer, amount, total);
        uint128 escrow = total - amount;

        env e_warn;
        require e_warn.msg.sender == funder;

        sinvoke warn(e_warn, signer);

        env e_pull;
        bool autolock;
        sinvoke pull(e_pull, signer, funder, autolock, amount, escrow);

        env e_post;
        uint256 funder_balance_post = sinvoke harness_token_balanceOf(e_post, funder);

        assert funder_balance_pre == funder_balance_post;
}
```

token.cvl

```
methods {
    totalSupply() returns uint256 envfree
}

invariant total_supply_initial()  sinvoke totalSupply() ==
1000000000000000000000000000
```

directory.cvl

```
methods {
      harness_get_pendings_amount(address,uint256) returns uint256 envfree
      harness_get_pendings_stakee(address,uint256) returns address envfree
      harness_get_delay(bytes32) returns uint128 envfree
      harness_get_amount(bytes32) returns uint256 envfree
      harness_contract_owner() returns address envfree
      pull(address,uint256,uint256)
      stop(uint256,uint256, uint128)
      take(uint256,uint256,address)
      push(address,uint256,uint128)
}

rule amount_zero(method f, address node) {
      env _e;
      env eF;
      env e_;

      require f != pull && f != push && f != stop;  // pull is expected to reduce
amount; stop is handled separately below

      uint128 _amount = sinvoke heft(_e,node);

      calldataarg arg;
      sinvoke f(eF,arg);

      uint128 amount_ = sinvoke heft(e_,node);

      assert _amount != 0 => amount_ != 0, "Function changes amount to zero";
}
```

```
rule amount_zero_push(address node, uint256 push_amount, uint128 delay) {
      env _e;
      env eF;
      env e_;

      uint128 _amount = sinvoke heft(_e,node);
      uint128 max_uint128 = 340282366920938463463374607431768211455;
      uint256 max_uint256 =
115792089237316195423570985008687907853269984665640564039457584007913129639935;

      require _amount + push_amount < max_uint256;

      sinvoke push(eF, node, push_amount, delay);

      uint128 amount_ = sinvoke heft(e_,node);

      assert _amount != 0 => amount_ != 0, "Function changes amount to zero";
}

rule amount_zero_stop(address node, uint256 _index, uint128 _delay) {
      env _e;
      env eF;
      env e_;

      uint128 _amount = sinvoke heft(_e,node);
      uint128 max_uint128 = 340282366920938463463374607431768211455;
      uint256 max_uint256 =
115792089237316195423570985008687907853269984665640564039457584007913129639935;

      // additional requirements
      uint256 pending = sinvoke harness_get_pendings_amount(eF.msg.sender,_index);
      // if pending + _amount overflows, we have a problem.
      require pending + _amount < max_uint256; // remove this line to get a violation

      uint256 stop_amount;
      sinvoke stop(eF, _index, stop_amount, _delay);

      uint128 amount_ = sinvoke heft(e_,node);

      assert _amount != 0 => amount_ != 0, "Function changes amount to zero";
}
```

```
// this rule is a query: it highlights which public functions can decrease delay
// we expect it to highlight only pull()
rule can_reduce_delay(method f,bytes32 key) {
	env _e;
	env eF;
	env e_;

	uint128 _delay = sinvoke harness_get_delay(key);

	calldataarg arg;
	sinvoke f(eF,arg);

	uint128 delay_ = sinvoke harness_get_delay(key);

	assert delay_ >= _delay, "Delay was decreased";
	//assert false,"sanity";
}

rule wait_revert_characterization(address stakee, uint128 delay) {
	uint128 max_uint128 = 340282366920938463463374607431768211455;

	env e_pre; env e_f;
	address staker = e_f.msg.sender;

	require 0 <= delay && delay <= max_uint128;

	bytes32 key = sinvoke name(e_pre, staker, stakee);
	uint256 before;
	uint256 after;
	uint256 amount;
	uint128 stake_delay;
	address stake_stakee;
	bytes32 parent;
	bytes32 left;
	bytes32 right;
	before, after, amount, stake_delay, stake_stakee, parent, left, right = sinvoke
harness_get_stake(e_pre, key);
	require 0 <= stake_delay && stake_delay <= max_uint128;

	require e_f.msg.value == 0;
```

```
        invoke wait(e_f, stakee, delay);
        bool wait_reverted = lastReverted;

        assert wait_reverted => (
                !(delay >= stake_delay) ||
                !(amount != 0)
        );
        //assert false,"sanity";
}

rule pull_take_delay(address stakee, uint128 amount, uint256 index, address target) {
        uint128 max_uint128 = 340282366920938463463374607431768211455;

        env e_pre;
        env e_pull;
        env e_take;

        address staker = e_pull.msg.sender;

        bytes32 key = sinvoke name(e_pre, staker, stakee);
        uint128 stake_delay = sinvoke harness_get_delay(key);

        calldataarg arg;
        sinvoke pull(e_pull, stakee, amount, index);
        bool pull_succeeded = !lastReverted;

        sinvoke take(e_take, index, amount, target);
        bool take_succeeded = !lastReverted;

        require e_pull.block.timestamp+stake_delay <= max_uint128; // we assume no
overflow in times.
        require e_pull.msg.sender == e_take.msg.sender;
        require e_take.block.timestamp >= e_pull.block.timestamp; // make
counterexamples more friendly

        assert (take_succeeded && pull_succeeded) => e_pull.block.timestamp +
stake_delay <= e_take.block.timestamp;
        //assert false,"sanity";
}

rule push_introduces_new_entry(address stakee, uint128 amount, uint128 delay) {
        require amount > 0;
        env e;
        address staker = e.msg.sender;
        bytes32 key = sinvoke name(e, staker, stakee);
        uint256 stake_pre_amount = sinvoke harness_get_amount(key);
        require stake_pre_amount == 0; // no entry at this key

        sinvoke push(e, stakee, amount, delay);

        uint256 stake_post_amount = sinvoke harness_get_amount(key);
        assert stake_post_amount > 0;
}

rule push_increases_amount(address stakee, uint256 amount, uint128 delay) {
        require amount > 0;

        env e;
        uint128 max_uint128 = 340282366920938463463374607431768211455;
        uint256 max_uint256 =
115792089237316195423570985008687907853269984665640564039457584007913129639935;
        address staker = e.msg.sender;
        bytes32 key = sinvoke name(e, staker, stakee);
        uint256 stake_pre_amount = sinvoke harness_get_amount(key);
        require stake_pre_amount + amount <= max_uint256; // assume no overflow
        require stake_pre_amount > 0; // no entry at this key

        sinvoke push(e, stakee, amount, delay);

        uint256 stake_post_amount = sinvoke harness_get_amount(key);
        assert stake_post_amount == stake_pre_amount + amount;
}

rule existing_stake_nonzero(address stakee, address staker, method f) {
        uint128 max_uint128 = 340282366920938463463374607431768211455;
        env e;

        require f != push && f != stop && f != pull;  // handled separately below

        bytes32 key = sinvoke name(e, staker, stakee);
        uint256 pre_before;
        uint256 pre_after;
        uint256 pre_amount;
        uint128 pre_stake_delay;
        address pre_stake_stakee;
        bytes32 pre_parent;
        bytes32 pre_left;
        bytes32 pre_right;
        pre_before,
        pre_after,
        pre_amount,
        pre_stake_delay,
        pre_stake_stakee,
        pre_parent,
        pre_left,
        pre_right = sinvoke harness_get_stake(e, key);

        require pre_amount == 0 => (pre_before == 0 && pre_after == 0 &&
pre_stake_delay == 0 && pre_stake_stakee == 0 && pre_parent == 0 && pre_left == 0 &&
pre_right == 0);

        env eF;
        calldataarg arg;
        sinvoke f(eF,arg);

        uint256 post_before;
        uint256 post_after;
        uint256 post_amount;
        uint128 post_stake_delay;
        address post_stake_stakee;
        bytes32 post_parent;
        bytes32 post_left;
        bytes32 post_right;
        post_before,
        post_after,
        post_amount,
        post_stake_delay,
        post_stake_stakee,
        post_parent,
        post_left,
        post_right = sinvoke harness_get_stake(e, key);

        assert  post_amount == 0 => (post_before == 0 && post_after == 0 &&
post_stake_delay == 0 && post_stake_stakee == 0 && post_parent == 0 && post_left == 0
&& post_right == 0);
```

```
}

rule existing_stake_nonzero_push(address stakee, address staker) {
	uint256 total_supply = 1000000000000000000000000000;
	env e;

	bytes32 key = sinvoke name(e, staker, stakee);
	uint256 pre_before;
	uint256 pre_after;
	uint256 pre_amount;
	uint128 pre_stake_delay;
	address pre_stake_stakee;
	bytes32 pre_parent;
	bytes32 pre_left;
	bytes32 pre_right;
	pre_before,
	pre_after,
	pre_amount,
	pre_stake_delay,
	pre_stake_stakee,
	pre_parent,
	pre_left,
	pre_right = sinvoke harness_get_stake(e, key);

	require pre_amount == 0 => (pre_before == 0 && pre_after == 0 &&
pre_stake_delay == 0 && pre_stake_stakee == 0 && pre_parent == 0 && pre_left == 0 &&
pre_right == 0);

	env eF;
	address push_staker = eF.msg.sender;
	address push_stakee;
	bytes32 push_key = sinvoke name(eF, push_staker, push_stakee);
	uint256 pre_push_amount = sinvoke harness_get_amount(push_key);
	uint256 push_amount;
	require push_amount + pre_push_amount <= total_supply;
	uint128 push_delay;
    // tree invariant: amount nonzero implies parent amount nonzero
	require (sinvoke harness_get_parent(eF, push_key)) == key && pre_push_amount !=
0 => pre_amount != 0;
	sinvoke push(eF, push_stakee, push_amount, push_delay);

	uint256 post_before;
```

```
        uint256 post_after;
        uint256 post_amount;
        uint128 post_stake_delay;
        address post_stake_stakee;
        bytes32 post_parent;
        bytes32 post_left;
        bytes32 post_right;
        post_before,
        post_after,
        post_amount,
        post_stake_delay,
        post_stake_stakee,
        post_parent,
        post_left,
        post_right = sinvoke harness_get_stake(e, key);

        assert post_amount == 0 => (post_before == 0 && post_after == 0 &&
post_stake_delay == 0 && post_stake_stakee == 0 && post_parent == 0 && post_left == 0
&& post_right == 0);
}

rule existing_stake_nonzero_stop(address stakee, address staker) {
        uint256 total_supply = 1000000000000000000000000000;
        env e;

        bytes32 key = sinvoke name(e, staker, stakee);
        uint256 pre_before;
        uint256 pre_after;
        uint256 pre_amount;
        uint128 pre_stake_delay;
        address pre_stake_stakee;
        bytes32 pre_parent;
        bytes32 pre_left;
        bytes32 pre_right;
        pre_before,
        pre_after,
        pre_amount,
        pre_stake_delay,
        pre_stake_stakee,
        pre_parent,
        pre_left,
        pre_right = sinvoke harness_get_stake(e, key);
        require pre_amount == 0 => (pre_before == 0 && pre_after == 0 &&
pre_stake_delay == 0 && pre_stake_stakee == 0 && pre_parent == 0 && pre_left == 0 &&
pre_right == 0);

        env eF;
        address stop_staker = eF.msg.sender;
        uint256 stop_index;
        uint256 stop_amount;
        uint128 stop_delay;
        address stop_stakee = sinvoke harness_get_pendings_stakee(stop_staker,
stop_index);
        uint256 stop_pending_amount = sinvoke harness_get_pendings_amount(stop_staker,
stop_index);
        bytes32 stop_key = sinvoke name(eF, stop_staker, stop_stakee);
        uint256 pre_stop_amount = sinvoke harness_get_amount(stop_key);
        require pre_amount + pre_stop_amount + stop_pending_amount <= total_supply;
    // tree invariant: amount nonzero implies parent amount nonzero
        require (sinvoke harness_get_parent(eF, stop_key)) == key && pre_stop_amount !=
0 => pre_amount != 0;

        sinvoke stop(eF, stop_index, stop_amount, stop_delay);

        uint256 post_before;
        uint256 post_after;
        uint256 post_amount;
        uint128 post_stake_delay;
        address post_stake_stakee;
        bytes32 post_parent;
        bytes32 post_left;
        bytes32 post_right;
        post_before,
        post_after,
        post_amount,
        post_stake_delay,
        post_stake_stakee,
        post_parent,
        post_left,
        post_right = sinvoke harness_get_stake(e, key);
```

```
	assert post_amount == 0 => (post_before == 0 && post_after == 0 &&
post_stake_delay == 0 && post_stake_stakee == 0 && post_parent == 0 && post_left == 0
&& post_right == 0);
}

rule existing_stake_nonzero_pull(address stakee, address staker) {
	uint256 total_supply = 1000000000000000000000000000;
	env e;

	bytes32 key = sinvoke name(e, staker, stakee);
	uint256 pre_before;
	uint256 pre_after;
	uint256 pre_amount;
	uint128 pre_stake_delay;
	address pre_stake_stakee;
	bytes32 pre_parent;
	bytes32 pre_left;
	bytes32 pre_right;
	pre_before,
	pre_after,
	pre_amount,
	pre_stake_delay,
	pre_stake_stakee,
	pre_parent,
	pre_left,
	pre_right = sinvoke harness_get_stake(e, key);

	require pre_amount == 0 => (pre_before == 0 && pre_after == 0 &&
pre_stake_delay == 0 && pre_stake_stakee == 0 && pre_parent == 0 && pre_left == 0 &&
pre_right == 0);

	env eF;
	address pull_staker = eF.msg.sender;
	address pull_stakee;
	uint256 pull_amount;
	uint256 pull_index;
	bytes32 pull_key = sinvoke name(eF, pull_staker, pull_stakee);
	uint256 pre_pull_amount = sinvoke harness_get_amount(pull_key);
	uint256 pull_pending_amount = sinvoke harness_get_pendings_amount(pull_staker,
pull_index);
	require pre_amount + pre_pull_amount + pull_pending_amount <= total_supply;
	address pull_parent = sinvoke harness_get_parent(eF, pull_key);
```

```
    // tree invariant: amount nonzero implies parent amount nonzero
        require pull_parent == key && pre_pull_amount != 0 => pre_amount != 0;

        // tree invariants: the children of n have parent fields pointing to n
        bytes32 pull_left = sinvoke harness_get_left(eF, pull_key);
        require pull_left == key => pre_parent == pull_key;
        address pull_right = sinvoke harness_get_right(eF, pull_key);
        require pull_right == key => pre_parent == pull_key;

        sinvoke pull(eF, pull_stakee, pull_amount, pull_index);

        uint256 post_before;
        uint256 post_after;
        uint256 post_amount;
        uint128 post_stake_delay;
        address post_stake_stakee;
        bytes32 post_parent;
        bytes32 post_left;
        bytes32 post_right;
        post_before,
        post_after,
        post_amount,
        post_stake_delay,
        post_stake_stakee,
        post_parent,
        post_left,
        post_right = sinvoke harness_get_stake(e, key);

        assert post_amount == 0 => (post_before == 0 && post_after == 0 &&
post_stake_delay == 0 && post_stake_stakee == 0 && post_parent == 0 && post_left == 0
&& post_right == 0);

}


rule only_pull_removes(address stakee, address staker, method f) {
        env e;
        uint128 total_supply = 1000000000000000000000000000;

        address owner = sinvoke harness_contract_owner();
        require sinvoke harness_token_balanceOf(e, owner) <= total_supply;
        bytes32 key = sinvoke name(e, staker, stakee);
        uint256 stake_pre_amount = sinvoke harness_get_amount(key);
        require stake_pre_amount > 0 && stake_pre_amount <= total_supply;
        require f != pull && f != stop; // stop handled in stop_does_not_remove

        env eF;
        calldataarg arg;
        sinvoke f(eF,arg);

        uint256 stake_post_amount = sinvoke harness_get_amount(key);
        require stake_post_amount <= total_supply;
        require sinvoke harness_token_balanceOf(e, owner) <= total_supply;
        assert stake_post_amount > 0;
}

rule stop_does_not_remove(uint256 index, uint256 amount, uint128 delay) {
        env e;
        uint128 total_supply = 1000000000000000000000000000;

        address staker = e.msg.sender;
        address stakee = sinvoke harness_get_pendings_stakee(staker, index);
        address owner = sinvoke harness_contract_owner();
        bytes32 key = sinvoke name(e, staker, stakee);
        uint256 stake_pre_amount = sinvoke harness_get_amount(key);

        require staker != owner && stakee != owner;
        require sinvoke harness_token_balanceOf(e, owner) <= total_supply;
        require stake_pre_amount > 0 && stake_pre_amount <= total_supply;
        require sinvoke harness_get_pendings_amount(staker, index) <= total_supply;

        sinvoke stop(e, index, amount, delay);

        uint256 stake_post_amount = sinvoke harness_get_amount(key);
        require stake_post_amount <= total_supply;
        require sinvoke harness_token_balanceOf(e, owner) <= total_supply;
        assert stake_post_amount > 0;
}

rule removed_does_not_exist(address stakee, address staker) {
        env e;

        bytes32 key = sinvoke name(e, staker, stakee);
        uint256 stake_pre_amount = sinvoke harness_get_amount(key);
        require stake_pre_amount > 0;

        uint256 index;

        require e.msg.sender == staker;
        sinvoke pull(e, stakee, stake_pre_amount, index);

        uint256 stake_post_amount = sinvoke harness_get_amount(key);
        assert stake_post_amount == 0;
}

rule stakees_mirrors_stakes(address stakee, address staker, method f) {
        env e;
        uint128 max_uint128 = 340282366920938463463374607431768211455;
        uint128 total_supply = 1000000000000000000000000000;
        bytes32 key = sinvoke name(e, staker, stakee);
        uint256 stake_pre_amount = sinvoke harness_get_amount(key);
        uint128 stakee_pre_amount = sinvoke heft(e, stakee);
        require stake_pre_amount <= stakee_pre_amount;

        calldataarg arg;
        sinvoke f(e, arg);

        uint256 stake_post_amount = sinvoke harness_get_amount(key);
        uint128 stakee_post_amount = sinvoke heft(e, stakee);

        require stake_pre_amount <= total_supply && stake_post_amount <= total_supply
&&
                        stakee_pre_amount <= total_supply && stakee_post_amount <=
total_supply;

        require stake_post_amount < stake_pre_amount && stakee_post_amount <
stakee_pre_amount;
        assert stake_post_amount - stake_pre_amount == stakee_post_amount -
stakee_pre_amount;
        // assert stake_post_amount != stake_pre_amount => (stake_pre_amount -
stake_post_amount == stakee_pre_amount - stakee_post_amount);
        // assert stake_post_amount < stake_pre_amount => stakee_post_amount <
stakee_pre_amount;
        // assert stake_post_amount > stake_pre_amount => stakee_post_amount >
stakee_pre_amount;
```

```
        // assert (stake_post_amount < stake_pre_amount && stakee_post_amount <
stakee_pre_amount) =>
        //              stake_post_amount - stake_pre_amount == stakee_post_amount -
stakee_pre_amount;
}

rule balance_mirrors_stakees_stop(uint256 index, uint256 amount, uint128 delay) {
        env e;
        address staker = e.msg.sender;
        uint256 totalSupply = 1000000000000000000000000000;
        address owner = sinvoke harness_contract_owner();
        address stakee = sinvoke harness_get_pendings_stakee(staker, index);

        uint256 balance_pre = sinvoke harness_token_balanceOf(e, owner);
        uint128 stakee_pre = sinvoke heft(e, stakee);
        uint256 pendings_pre = sinvoke harness_get_pendings_amount(staker, index);

        sinvoke stop(e, index, amount, delay);

        uint256 balance_post = sinvoke harness_token_balanceOf(e, owner);
        uint128 stakee_post = sinvoke heft(e, stakee);
        uint256 pendings_post = sinvoke harness_get_pendings_amount(staker, index);

        require balance_pre <= totalSupply && balance_post <= totalSupply &&
                     stakee_pre + pendings_pre <= totalSupply && stakee_post +
pendings_post <= totalSupply;
        assert balance_post - balance_pre == (pendings_post + stakee_post) -
(pendings_pre + stakee_pre);
}

rule balance_mirrors_stakees_take(uint256 index, uint256 amount, address target) {
        env e;
        address staker = e.msg.sender;
        uint256 totalSupply = 1000000000000000000000000000;
        address owner = sinvoke harness_contract_owner();
        require target != owner;
        address stakee = sinvoke harness_get_pendings_stakee(staker, index);

        uint256 balance_pre = sinvoke harness_token_balanceOf(e, owner);
        uint128 stakee_pre = sinvoke heft(e, stakee);
        uint256 pendings_pre = sinvoke harness_get_pendings_amount(staker, index);
        sinvoke take(e, index, amount, target);

        uint256 balance_post = sinvoke harness_token_balanceOf(e, owner);
        uint128 stakee_post = sinvoke heft(e, stakee);
        uint256 pendings_post = sinvoke harness_get_pendings_amount(staker, index);

        require balance_pre <= totalSupply && balance_post <= totalSupply &&
                        stakee_pre + pendings_pre <= totalSupply && stakee_post +
pendings_post <= totalSupply;
        assert balance_post - balance_pre == (pendings_post + stakee_post) -
(pendings_pre + stakee_pre);
}

rule balance_mirrors_stakees_pull(uint256 index, uint128 amount) {
        env e;
        address staker = e.msg.sender;
        uint256 totalSupply = 1000000000000000000000000000;
        address owner = sinvoke harness_contract_owner();
        address stakee = sinvoke harness_get_pendings_stakee(staker, index);

        uint256 balance_pre = sinvoke harness_token_balanceOf(e, owner);
        uint128 stakee_pre = sinvoke heft(e, stakee);
        uint256 pendings_pre = sinvoke harness_get_pendings_amount(staker, index);

        sinvoke pull(e, stakee, amount, index);

        uint256 balance_post = sinvoke harness_token_balanceOf(e, owner);
        uint128 stakee_post = sinvoke heft(e, stakee);
        uint256 pendings_post = sinvoke harness_get_pendings_amount(staker, index);

        require balance_pre <= totalSupply && balance_post <= totalSupply &&
                        stakee_pre + pendings_pre <= totalSupply && stakee_post +
pendings_post <= totalSupply;
        assert balance_post - balance_pre == (pendings_post + stakee_post) -
(pendings_pre + stakee_pre);
}

rule balance_mirrors_stakees_push(uint256 index, uint128 amount, uint128 delay) {
        env e;
        address staker = e.msg.sender;
        uint256 totalSupply = 1000000000000000000000000000;
        address owner = sinvoke harness_contract_owner();
        address stakee = sinvoke harness_get_pendings_stakee(staker, index);

        uint256 balance_pre = sinvoke harness_token_balanceOf(e, owner);
        uint128 stakee_pre = sinvoke heft(e, stakee);
        uint256 pendings_pre = sinvoke harness_get_pendings_amount(staker, index);

        require staker != owner;

        sinvoke push(e, stakee, amount, delay);

        uint256 balance_post = sinvoke harness_token_balanceOf(e, owner);
        uint128 stakee_post = sinvoke heft(e, stakee);
        uint256 pendings_post = sinvoke harness_get_pendings_amount(staker, index);

        require balance_pre <= totalSupply && balance_post <= totalSupply &&
                     stakee_pre + pendings_pre <= totalSupply && stakee_post +
pendings_post <= totalSupply;
        assert balance_post - balance_pre == (pendings_post + stakee_post) -
(pendings_pre + stakee_pre);
}

rule balance_mirrors_stakees(address stakee, method f) {
        env e;
        uint256 totalSupply = 1000000000000000000000000000;
        address owner = sinvoke harness_contract_owner();

        uint256 balance_pre = sinvoke harness_token_balanceOf(e, owner);
        uint128 stakee_pre = sinvoke heft(e, stakee);

        // do not tests methods that have their own version of this rule
        require f != stop && f != take && f != pull && f != push;

        calldataarg arg;
        sinvoke f(e, arg);

        uint256 balance_post = sinvoke harness_token_balanceOf(e, owner);
        uint128 stakee_post = sinvoke heft(e, stakee);

        require balance_pre <= totalSupply && balance_post <= totalSupply &&
                     stakee_pre <= totalSupply && stakee_post <= totalSupply;

        assert balance_post - balance_pre == stakee_post - stakee_pre;
```

```
}

rule wellformed_tree_bounded() {
        env e;
        //sinvoke harness_unittest(e);
        uint256 totalSupply = 1000000000000000000000000000;
        address stakee;
        address staker = e.msg.sender;
        bytes32 key = sinvoke name(e, staker, stakee);
        uint256 index;

        bytes32 root = sinvoke harness_get_root(e);
        uint256 root_amount = sinvoke harness_get_amount(root);
        bytes32 left = sinvoke harness_get_left(e, root);
        bytes32 left_parent = sinvoke harness_get_parent(e, left);
        bytes32 left_left = sinvoke harness_get_left(e, left);
        bytes32 left_right = sinvoke harness_get_right(e, left);
        bytes32 right = sinvoke harness_get_right(e, root);
        bytes32 right_parent = sinvoke harness_get_parent(e, right);
        bytes32 right_left = sinvoke harness_get_left(e, right);
        bytes32 right_right = sinvoke harness_get_right(e, right);

        require root != 0 && left != 0 && right != 0;
        require root == key;
        require left_parent == root && right_parent == root;
        require left != right && left != root && right != root;
        require left_left == 0 && left_right == 0 && right_left == 0 && right_right ==
0;
        require 0 < root_amount && root_amount <= totalSupply;

        invoke pull(e, stakee, root_amount, index);
        assert !lastHasThrown;

        bytes32 root_post = sinvoke harness_get_root(e);
        uint256 root_amount_post = sinvoke harness_get_amount(root_post);
        bytes32 left_post = sinvoke harness_get_left(e, root_post);
        bytes32 right_post = sinvoke harness_get_right(e, root_post);

        require 0 < root_amount_post && root_amount_post <= totalSupply;

        assert left_post != root_post && right_post != root_post;
        assert left_post != 0 || right_post != 0;
```

```
}
```

curator.cvl

```
rule owner_unchanged(method f) {
	env e_pre;
	address owner_pre = sinvoke jrw_get_owner(e_pre);

	env e;
	calldataarg a;
	sinvoke f(e, a);

	env e_post;
	address owner_post = sinvoke jrw_get_owner(e_post);

	assert owner_post == owner_pre;
}


rule curated_unchanged(method f) {
	address target;

	env e_pre;
	bool good_pre = sinvoke good(e_pre, target);
	address owner = sinvoke jrw_get_owner(e_pre);

	env e;
	calldataarg a;
	sinvoke f(e, a);

	env e_post;
	bool good_post = sinvoke good(e_post, target);

	assert good_pre != good_post => e.msg.sender == owner;
}
```