

# CVL: Basic Rules



**Michael George**

**Stanford, August 2022**

# Overview for this session

## Basic rules for ERC20 contracts

- ▶ Presentation: writing and debugging rules
  - ▶ transfer changes balances appropriately
  - ▶ transfer reverts when it should
  - ▶ transfer doesn't revert unexpectedly
- ▶ Exercise: similar rules for transferFrom

## Generalized (parametric) rules

- ▶ Presentation: rules that apply to all methods
  - ▶ Only the owner can increase their allowance
  - ▶ The owner only changes their allowance deliberately
- ▶ Exercise: similar rules for balanceOf

# ERC20 transfer and balanceOf

The first properties we'd like to test are described in the interface:

```
//// contracts/IERC20.spec

/**
 * Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {

    /**
     * Moves `amount` tokens from the caller's account to `recipient`.
     */
    function transfer(address recipient, uint256 amount)
        external
        returns(bool);

    /**
     * Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account)
        external view
        returns(uint256);

    ...
}
```

# Specifying transfer in CVL (unit-test-style rules)

```
//// certora/specs/ERC20.spec

methods {
  balanceOf(address) returns (uint) envfree
}

/// Transfer must move `amount` tokens from
/// the caller's account to `recipient`.
rule transferSpec {
  address sender; address recip; uint amount;

  env e;
  require e.msg.sender == sender;

  mathint balance_sender_before = balanceOf(sender);
  mathint balance_recip_before  = balanceOf(recip);

  transfer(e, recip, amount);

  mathint balance_sender_after = balanceOf(sender);
  mathint balance_recip_after  = balanceOf(recip);

  require sender != recip;

  assert balance_sender_after == balance_sender_before - amount,
    "transfer must decrease sender's balance by amount";

  assert balance_recip_after  == balance_recip_before  + amount,
    "transfer must increase recipient's balance by amount";
}
```

(results link)

(second link)

## What about revert?

So far:

- ▶ Transfer reduces sender's balance by amount
- ▶ Transfer increases recipient's balance by amount

What if sender's balance is less than amount?

- ▶ Transaction reverts
- ▶ No balances change!
- ▶ Why doesn't this violate the rule?

Answer: by default, Prover ignores reverting paths.

- ▶ we can override this behavior to reason about reverting

# transfer revert conditions

```
//// certora/specs/ERC20.spec

/// Transfer must revert if the sender's balance is too small
rule transferReverts {
  env e; address recip; uint amount;

  require balanceOf(e.msg.sender) < amount;

  transfer@withrevert(e, recip, amount);

  assert lastReverted,
    "transfer(recip,amount) must revert if sender's balance is less than `amount`";
}
```

Results	Contract list
<input type="text" value="Type to filter"/>	All results <input type="button" value="🗉"/>
✔ transferReverts	0s

[\(results link\)](#)

## Reasoning about reverts:

- ▶ Use `f@withrevert(...)` to consider paths where `f` reverts
- ▶ Use `lastReverted` to determine whether last call reverted
  - ▶ **Warning:** it is always the last call!
  - ▶ save it if you need to make another call

# Proving transfer doesn't revert (liveness rules)

```
//// certora/specs/ERC20.spec

/// Transfer must not revert unless
///   - the sender doesn't have enough funds
///
///
///
///
/// @title Transfer doesn't revert
rule transferDoesntRevert {
  env e; address recipient; uint amount;

  require balanceOf(e.msg.sender) > amount;

  transfer@withrevert(e, recipient, amount);
  assert !lastReverted;
}
```

## Call Trace



multi contract setup

rule parameters setup

last storage initialize

assumptions about extcodesize

assumptions about starting balances

record starting nonces

cloned contracts have no balances

Linked immutable setup

> require balanceOf(e.msg.sender) > amount

▼ transfer(e,recipient,amount) could\_revert

└─ ERC20.transfer(recipient=0x401 (same as recipient), amount=13)

REVERT

└─ Why did this call revert?()

REVERT CAUSE

└─ See "\\*contract ERC20 is IERC20, IERC20Metadata {...}" @  
certora\_config/autoFinder\_ERC20.sol\_0/2\_autoFinder\_ERC20.sol: line 34()

└─ !(e.msg.value==0x0)()

DUMP

> assert !lastReverted

# Proving transfer doesn't revert (liveness rules)

```
//// certora/specs/ERC20.spec

/// Transfer must not revert unless
///   - the sender doesn't have enough funds
///   - or the message value is nonzero,
///
///
///
/// @title Transfer doesn't revert
rule transferDoesntRevert {
  env e; address recipient; uint amount;

  require balanceOf(e.msg.sender) > amount;
  require e.msg.value == 0;

  transfer@withrevert(e, recipient, amount);
  assert !lastReverted;
}
```

```
> require balanceOf(e.msg.sender) > amount
> require e.msg.value == 0
v transfer(e,recipient,amount) could_revert
└ v ERC20.transfer(recipient=0xffff (same as recipient), amount=2) REVERT
  └ (internal) ERC20.transfer(recipient=0xffff (same as recipient), amount=2)
    └ (internal) ERC20._transfer(sender=0xfffe (same as e.msg.sender),
      recipient=0xffff (same as recipient), amount=2) REVERT
      └ (internal) ERC20._beforeTokenTransfer(from=0xfffe (same as
        e.msg.sender), to=0xffff (same as recipient), amount=2)
        > Load from _balances[*]: 15
        > Store at _balances[*]: 13
        > Load from _balances[*]: 0xfffffffffffffffffffffffffffffffffffffe
        v Why did this call revert?() REVERT CAUSE
          └ W137[R141]>((0x2^0x100 -int 0x1)-amount)() DUMP

> assert !lastReverted
```



# Proving transfer doesn't revert (liveness rules)

```
//// certora/specs/ERC20.spec

/// Transfer must not revert unless
///   - the sender doesn't have enough funds
///   - or the message value is nonzero,
///   - or the recipient's balance would overflow,
///
///
///
/// @title Transfer doesn't revert
rule transferDoesntRevert {
  env e; address recipient; uint amount;

  require balanceOf(e.msg.sender) > amount;
  require e.msg.value == 0;
  require balanceOf(recipient) + amount < max_uint;

  transfer@withrevert(e, recipient, amount);
  assert !lastReverted;
}
```

```
> require balanceOf(e.msg.sender) > amount
> require e.msg.value == 0
> require balanceOf(recipient)+intamount < max_uint
v transfer(e,recipient,amount) could_revert
└ v ERC20.transfer(recipient=0x2711 (same as recipient), amount=2) REVERT
  └ v (internal) ERC20.transfer(recipient=0x2711 (same as recipient), amount=2)
    └ > (internal) ERC20._transfer(sender=0x0 (same as e.msg.sender),
      recipient=0x2711 (same as recipient), amount=2) REVERT

> assert !lastReverted
```

# Proving transfer doesn't revert (liveness rules)

```
//// certora/specs/ERC20.spec

/// Transfer must not revert unless
///   - the sender doesn't have enough funds
///   - or the message value is nonzero,
///   - or the recipient's balance would overflow,
///   - or the message sender is 0
///
///
/// @title Transfer doesn't revert
rule transferDoesntRevert {
  env e; address recipient; uint amount;

  require balanceOf(e.msg.sender) > amount;
  require e.msg.value == 0;
  require balanceOf(recipient) + amount < max_uint;
  require e.msg.sender != 0;

  transfer@withrevert(e, recipient, amount);
  assert !lastReverted;
}
```

```
> require balanceOf(e.msg.sender) > amount
> require e.msg.value == 0
> require balanceOf(recipient)+intamount < max_uint
> require e.msg.sender != 0
v transfer(e,recipient,amount) could_revert
└─ v ERC20.transfer(recipient=0x0 (same as recipient), amount=13) REVERT
    └─ v (internal) ERC20.transfer(recipient=0x0 (same as recipient), amount=13)
        └─ > (internal) ERC20._transfer(sender=0x2711 (same as
            e.msg.sender), recipient=0x0 (same as recipient), amount=13) REVERT

> assert !lastReverted
```

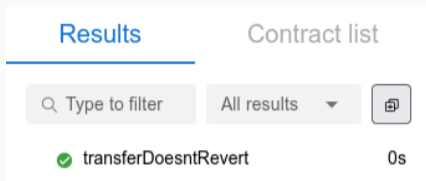
# Proving transfer doesn't revert (liveness rules)

```
//// certora/specs/ERC20.spec

/// Transfer must not revert unless
///   - the sender doesn't have enough funds
///   - or the message value is nonzero,
///   - or the recipient's balance would overflow,
///   - or the message sender is 0
///   - or the recipient is 0
///
/// @title Transfer doesn't revert
rule transferDoesntRevert {
  env e; address recipient; uint amount;

  require balanceOf(e.msg.sender) > amount;
  require e.msg.value == 0;
  require balanceOf(recipient) + amount < max_uint;
  require e.msg.sender != 0;
  require recipient != 0;

  transfer@withrevert(e, recipient, amount);
  assert !lastReverted;
}
```



The screenshot shows a web interface with two tabs: "Results" (active) and "Contract list". Below the tabs is a search bar with the placeholder text "Type to filter". To the right of the search bar is a dropdown menu currently set to "All results" and a copy icon. Below these elements, a single result is displayed: a green checkmark icon followed by the text "transferDoesntRevert" and "0s" to its right.

[\(results link\)](#)

## Summary

- ▶ Writing rules is like writing unit tests
  - ▶ But you can let the prover choose the values!
- ▶ Use `mathint` variables to avoid overflow in spec
- ▶ Pass `env` as first argument to specify `msg.sender` and other variables
  - ▶ Use `envfree` declaration in `methods` block to avoid passing `env`
- ▶ By default, reverting paths are ignored
  - ▶ Use `@withrevert` and `lastReverted` to reason about reverting paths
  - ▶ Writing “liveness properties” is hard (but possible!)

## Exercise (~15 minutes)

So far (certora/specs/ERC20.spec):

- ▶ transferSpec
- ▶ transferReverts
- ▶ transferDoesntRevert

Exercise:

- ▶ Write transferFromSpec
  - ▶ ...get it to pass
- ▶ Try transferFromReverts
- ▶ Try transferFromSucceeds

To run:

```
sh certora/scripts/verifyERC20.sh
```

Ask for help!

```
//// contracts/IERC20.sol

/// Interface of the ERC20 standard as defined in the EIP.
interface IERC20 {

    /// Moves `amount` tokens from `sender` to `recipient` using
    /// the allowance mechanism. `amount` is then deducted from
    /// the caller's allowance.
    ///
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);

    /// Returns the remaining number of tokens that `spender`
    /// will be allowed to spend on behalf of `owner` through
    /// {transferFrom}.
    ///
    /// This value changes when {approve} or {transferFrom} are
    /// called.
    ///
    function allowance(address owner, address spender)
        external
        view
        returns(uint256);
}
```