The Art and Science of Designing Specifications

certora

Michael George

Stanford, August 2022

certora

Types of properties

So far: how to write specs

Now: what specs to write?

When designing specifications, it helps to work systematically

- Unit-test-style rules
- Variable relationships and changes
- State transition diagrams
- Stakeholder rules
- High-level properties

Unit-test style rules

- Public functions and interfaces should have documentation
  - Describe what their arguments are
  - Describe what effects they should have
  - Describe what they should return
  - Describe when they should revert
- This documentation can usually be turned directly into specs
  - You can write one or more rules for each method
  - We call these “unit-test style rules”
  - Example: transfer decreases sender’s balance by amount
- Note: you can get a list of public functions from the Prover (example)
- In practice, the documentation is often incomplete
  - Think about the documentation you’d write
  - Maybe submit a PR!

Variable relationships and changes

Variable relationships

- For each pair of variables, ask “how are they related”?
- Each relationship can be written as an invariant
- Include related contracts!

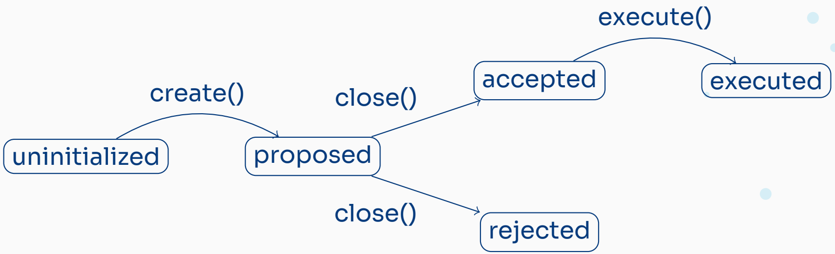
Variable changes

- For each variable, ask “how can it change, and when?”
- Each variable has one or more parametric rules:

```
rule variableChange {
    mathint value_before = getValue();

    method f; env e; calldataarg args;
    f(e,args);

    mathint value_after = getValue();

    assert value_before != value_after => ...;
}
```

State transition diagrams

Often contracts have a natural “flow-chart” feel:



These can naturally be turned into rules:

- Define properties of each state

```
definition accepted_state (env e) returns bool =
    initialized() && executable() != 0 && for() > against() && e.block.timestamp > deadline()
```

- Invariant: contract is always in one and only one state
- Each transition can have one or more rules, like variable changes

Stakeholder rules

Think about what can go wrong from stakeholders' perspectives

- User: I deposit funds and can't get them back
- Bank: Someone removes all the funds

Each "user (horror) story" can be turned into properties

Often multiple rules: e.g. to show "after deposit I can reclaim funds"

- If I deposit, I get a balance
- My balance doesn't go down unless I withdraw or transfer
- I can always withdraw without revert
- When I withdraw, the contract transfers tokens to me

High-level properties

There are some simple properties that can often get good coverage

- If this goes up, that goes up (correlation)
- If this is zero, that is zero
- Two small operations are the same as one big operation (additivity)
- Different ways to do the same thing have the same effect

Sometimes, more abstract properties are useful

- Get good coverage quickly
- Help us think in a different way, avoiding spec bugs

certora

Summary

When designing specifications, it helps to work systematically

- Unit-test-style rules
  - Describe the expected behavior of each function
- Variable relationships and changes
  - Describe the relationships between pairs of variables
  - Describe the conditions when variables change
- State transition diagrams
  - Identify parts of the contract that transition from state to state
  - Check that contract is always in exactly one state
  - Describe conditions when transitions happen
- Stakeholder rules
  - Think about what can go wrong
  - Look at your advertising
- High-level properties
  - Think abstractly about your functions and their relationships

certora

AAVE Token Example

Voting and delegation

The AAVE token is used for voting on proposals

- The more tokens you hold, the more votes you get

You can delegate your vote to another address:

- Delegation is all-or-nothing
- You can’t redelegate tokens

Delegation:

$$\begin{tikzcd} \boxed{\text{alice}} \arrow[r, bend left] & \boxed{\text{bob}} \arrow[r, bend left] & \boxed{\text{chuck}} \arrow[l, bend left] \end{tikzcd}$$

Token balance:	10	7	5
Voting power:	0	15	7

certora

A few more details

- The token manages two types of voting power: `VOTING` and `PROPOSITION`
- The contract supports “meta-delegation”
  - Allows delegation for someone other than `msg.sender`
  - Requires a digital certificate
- The contract is also an ERC20

Exercise: write (English) properties for governance

1. Fetch the code: in the `Examples` repo,
   - `git pull`
   - `git submodule update --init`
   - Alternately, get directly at `https://github.com/Certora/aave-token-v3`
2. Review the interfaces
   - Main interface is in `src/interfaces/IGovernancePowerDelegationToken.sol`
   - The token also implements the ERC20 interface
3. Start writing down properties!
   - `https://bit.ly/certora-stanford/`