[NOTE]SCV-Case Study

Source : r0.py

大纲

- Case Study (r0/r1)
  - States Modeling
  - Induction on States
  - Encoding Property
  - Modeling Contract(roughtly)
  - **RESULT**
- 感受(部分)
- 其它

Case Study

目标性质

> The sum of all investor deposits equals the ether balance of the escrow unless the crowdsale is declared successful.

sum of all deposits : the real total deposits
ether balance of escrow : the computed total deposits

相关 States Modeling

- EVM
  - eth balances : `eth_balances = Array('eth_balances', AddressSort, UintSort)`
- investor deposits : `investor_deposits = Array('investor_deposits', AddressSort, UintSort)`
- raised : `raised = Const('raised', UintSort)`
- escrowstate : `escrowstate = Const('escrowstate', EscrowState)`

Induction on States

赋值集合 = Symbolic Values + Constraints
证明正确性的来源 = Inductive Proof

性质命题

```
P : sum of all investor deposits == eth_balance[Escrow_Address]
```

- BASE CASE : 存在一些状态(e.g. 合约一开始被创建时的状态), 如果corwdsale没有声明successful, 则P成立
- INDUCTIVE CASE : 对于任意的使性质P成立的状态, 如果...., 在经过任意一次transition后依然成立

model任意使性质P成立的状态(表示为无约束的符号变量):

- `raised`
- `investor_deposits`
- arbitary investor
- `escrowstate`

考虑
Zellic Blog:

> - **balanceOf = Store(balanceOf, myUser, Uint(0))**: The user starts out with no WETH before their deposit. (this is OK because WLOG we can always get to this “known safe state” in real life by simply withdrawing all our WETH)

(**或许**)参考上面的叙述 *arbitrary valid states* 都可以通过这种方法退回到invest为0的状态(但我觉得这件事对于形式化验证来说并不是trivial的), 所以我认为以下的假设中 `raised = eth_balance[Escrow_Address]` 可以代表任意性质P成立的状态空间.

正确假设都来自目标系统的spec/std/proposal.

相关约束与假设:

```
a = Const('a', AddressSort)
require(s, ForAll([a], ULE(investor_deposits[a], eth_balances[Escrow_Address])))
require(s, ForAll([a], ULE(eth_balances[a], MAX_ETH)))

# [ASSUMPTION]
# - ether balance of escrow and sum of all deposits is equall at the beginning
# , here assume both to be 0
# raised, eth_balance[escrow_address], all elem of deposits = 0
# - only one depositor
investor_deposits = Store(investor_deposits, myUser, Uint(0))
eth_balances = Store(eth_balances, Escrow_Address, Uint(0))
raised = eth_balances[Escrow_Address]
```

对性质命题encoding

> The sum of all investor deposits equals the ether balance of the escrow unless the crowdsale is declared successful.

=>

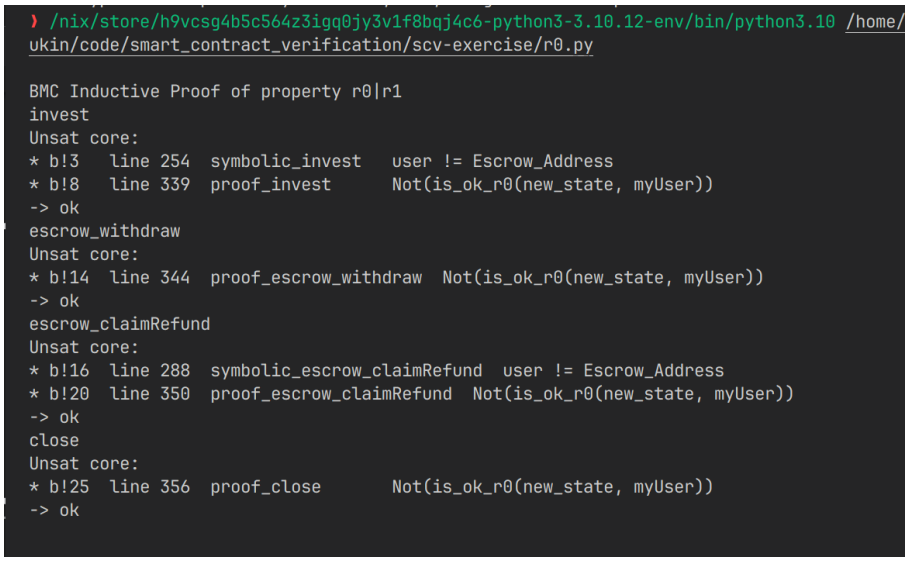
```
p = If(escrowstate != EscrowState.SUCCESS,
           raised == eth_balances[Escrow_Address],
           True)
```

Modeling Contract (概述)

- quantifier : side-effect为添加constraints的函数
- external call : symbolic execution
- functions : states trans rules

- Constructor : 实例化过程就是赋予其单独的状态空间, escrow实例化涉及的只有benificinary和owner, 所以只要把它们定义出来.
- time : 忽略, 无关状态

RESULT



invest和escrow_claimRefund的unsat core让验证看上去比较合理, 另外许多约束的作用是防止state explosion problem和排除一些与目标对象的性质不符的counter example.

感觉(部分)

Modeling是一个依赖于对目标系统(spec/standard/proposal)的熟悉程度和验证书写者经验的工作.
过程中我对sum of all sth用z3py书写感到很别扭, 而涉及这个东西的性质在智能合约里应该是很常见的?
所以我觉得可以有更好的工具.

其它

- 验证的正确性
  - 也许这里有一些能够降低验证门槛的方法?
    - 约束/假设依赖于目标系统的spec/标准
    - 测试技巧
      - 颅内模拟状态机
      - 加一些错误的约束
      - maybe more?
- 合理的假设/约束
  - 与验证工具的关系? 会因为工具不同吗.(我觉得是的)
  - 有没有一种更加General的方法**自动**得到全部或者部分的约束?
- 验证的可行性与效率
  - 避免Explosion
- 可以有更好的工具
  - 表达能力
  - 更自然的书写(语法/sum of sth)

- Faster