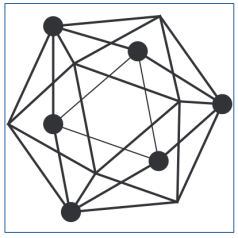
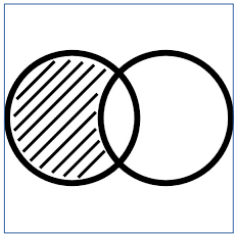
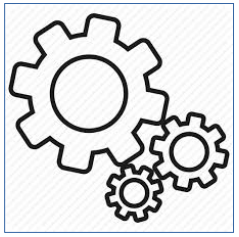
Семинар ТРАП

О проблеме повышения надёжности смарт-контрактов через символьную верификацию модели




Евгений Шишкин
ИнфоТеКС
2020

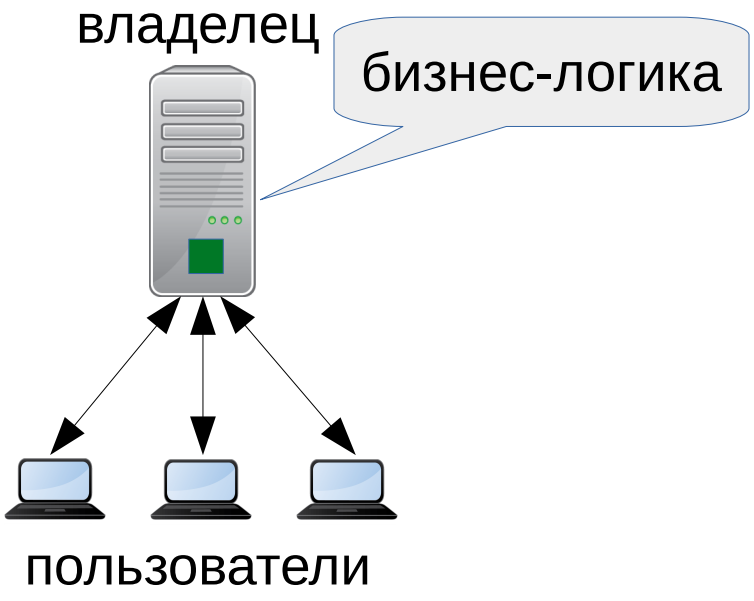
ИСП РАН

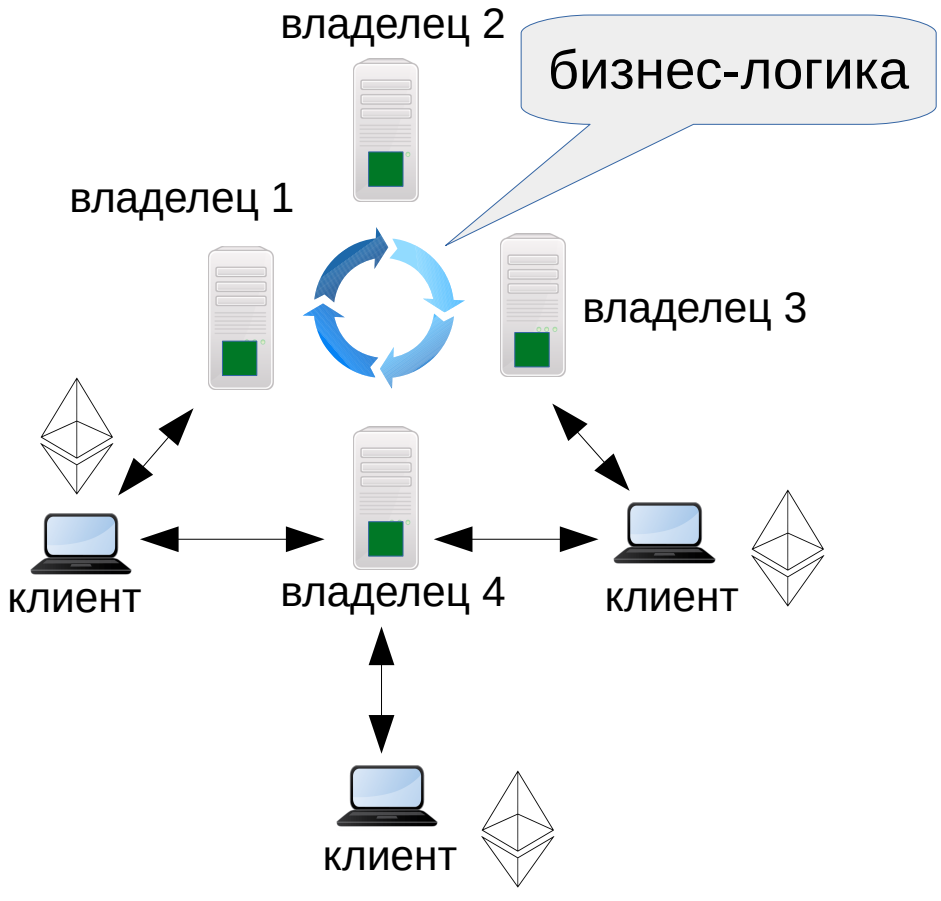
infotecs®

О докладе

- Какую проблему мы решаем?
- Зачем мы её решаем?
- Зачем нужен этот доклад?

Что такое блокчейн

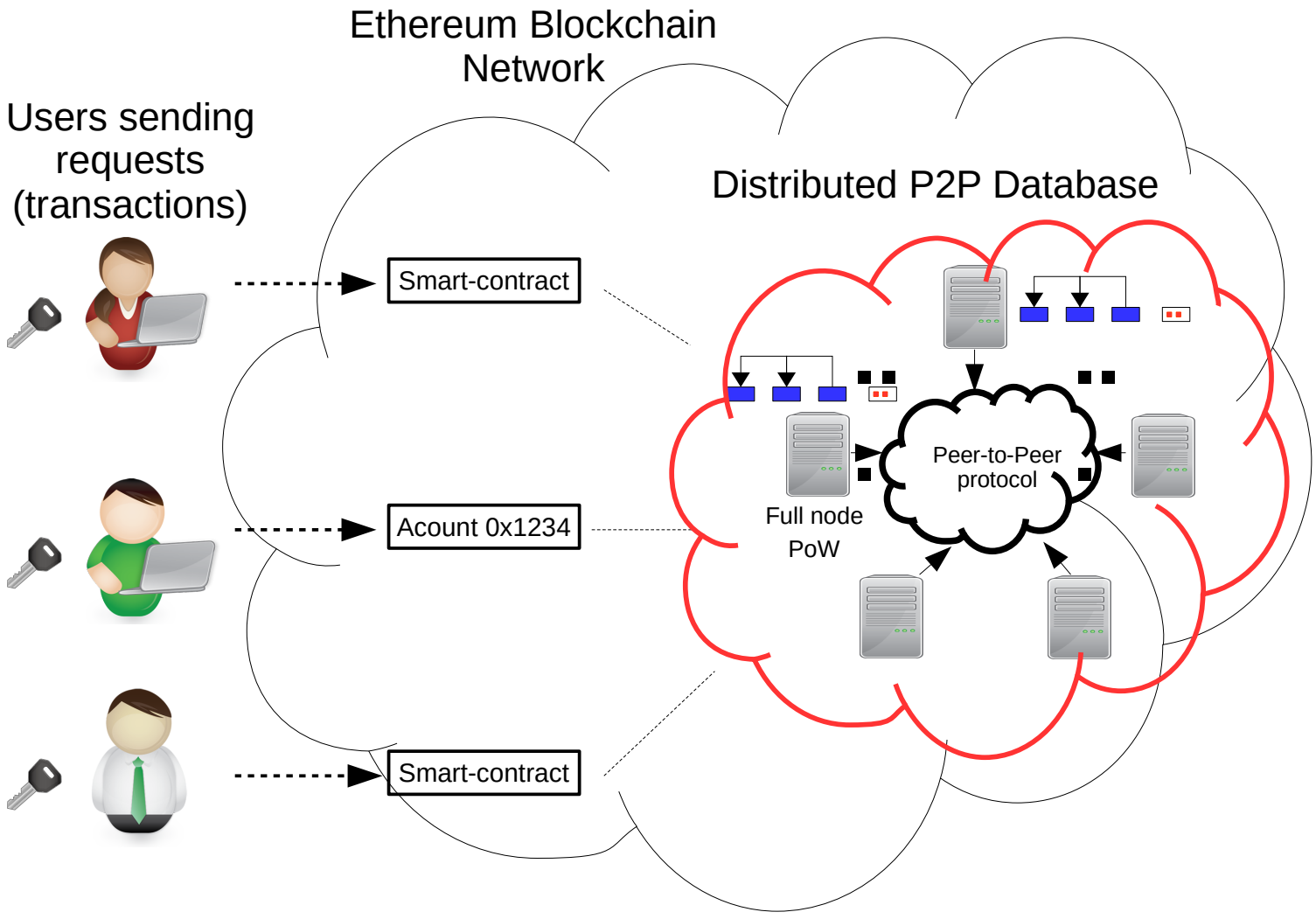




Ценность технологии

- Уменьшение транзакционных издержек
- Прозрачные неподменяемые вычисления

Архитектура Блокчейн-системы



Смарт-контракты Ethereum

1. (Квази) Тьюринг-полные программы для вирт. машины EVM
2. Реагирующие системы, состоящие из набора обработчиков входящих сообщений
3. Смарт-контракты могут взаимодействовать между собой и создавать новые контракты

4. **НО**: на практике
   - Смарт-контракты ограничены в своём функционале
   - Имеют схожую структуру

Языки Программирования для EVM

1. Yul - высокоуровневый ассемблер EVM со статической типизацией
2. Vyper - Python-подобный язык со строго завершающимися вычислениями
3. Solidity - JavaScript-подобный ООП язык

4. Скоро: 🔥 Formality[1] - Haskell-подобный язык с зависимыми типами и встроенной поддержкой построения доказательств

[1] https://github.com/moonad/Formality

Язык Solidity I

```
contract SampleToken is ERC20, Refundable {
  // Value type into Reference type mapping
  mapping (uint => mapping (address => uint)) tokens;

  // Multiple return values
  function calculateBonus(address benef, uint256 amount)
      internal returns (uint256 a, uint256 b) {
    //...
    return (1,2);
  }
}
```

Язык Solidity II

```
contract SampleToken is ERC20, Refundable {
  bool paused;

  modifier whenNotPaused {
    require (paused == false);
    _;
  }

  // Modifiers
  function transferFrom(
     address _from,
     address _to,
     uint256 _value) public whenNotPaused returns (bool)
  {
    //...
  }
}
```

Язык Solidity III

```
contract SampleToken is ERC20, Refundable {
  // Event records
  event Transfer(address from, address to, uint val);

  function transferFrom(
     address _from,
     address _to,
     uint256 _value) public whenNotPaused returns (bool)
  {
    // put event record into the log
    emit Transfer(_from, _to, _value);
  }
}
```

Язык Solidity IV

По мимо этого...

- Произвольные циклы **for**,**while**, рекурсия
- Вызов функций другого смарт-контракта
- Библиотеки
- Возможность обратного вызова
- Ограниченная глубина стека
- Исключения без возможности их перехвата
- Функциональный тип переменной
- Динамические контейнеры
- Различная семантика оператора присваивания
- Неявные параметры: время, балансы адресов

Пример уязвимости

```
contract BuyTokens {
  uint public tokenPrice;
  uint public bonus;
  mapping (address => uint) balances;
  // ... skipped ...
  constructor() public {
    owner = msg.sender;
  }

  function setBonus(uint newBonus) external onlyOwner {
    bonus = newBonus;
  }

  function setPrice(uint newPrice) external onlyOwner {
    tokenPrice = newPrice;
  }

  function buyTokens(uint tokens) external payable {
    uint tokens = bonus + msg.value / tokenPrice;
    balances[msg.sender] += tokens;
  }
}
```

Векторы атак на смарт-контракты

- Ошибки при компиляции в байткод
- Нехватка газа для очередной транзакции
- Операционные ошибки (переполнения, reentrancy, etc)
- Зависимость от порядка выполнения транзакций
- ...

- **Логические ошибки**
  - Алгоритм не во всех случаях достигает желаемого результата
  - Тупиковые состояния
  - Небезопасные состояния

Формальная верификация

Логическая ошибка - отклонение поведения системы от её *формальной спецификации*.

Задача формальной верификации в том, чтобы убедиться в отсутствии логических ошибок.

Через логический вывод:

$$Impl \Rightarrow Spec$$

$$(Impl \Rightarrow Inv) \wedge (Impl \wedge Inv \Rightarrow Spec)$$

Через проверку включения множеств состояний:

$$Impl \subseteq Spec \iff Impl \cap \overline{Spec} = \emptyset$$

Через построение отношения бисимуляции:

$$Impl_1 \approx_B Impl_2$$

Символьная верификация модели

1. Построение спецификации

$$WhitePaper \dashrightarrow \{Req_1, ..., Req_n\}$$

$$Spec = \bigwedge_i Req_i$$

2. Построение модели по программе и спецификации

$$Model_i = \pi(Impl, Req_i)$$

3. Проверка соответствия модели и спецификации

$$check(Impl, Spec) = \bigwedge_i check(Model_i, Req_i)$$

В качестве бэкэнда испольется SMT-решатель Z3.

Модель исполнения смарт-контракта

```
contract Sample {
  uint x;
  function f(uint a, uint b) external payable
    returns (uint) {
    // implicit arguments:
    //   contract state (i.e. x var), msg.sender,
    //   msg.value, block.timestamp, user balances
    // not available:
    //   log, alive
    //
  }
}
```

$$\sigma := \langle \sigma_c, b, t \rangle, \sigma_c := \langle \sigma_{cs}, alive, log \rangle, \sigma_{cs} : \mathbb{N} \to Val$$

$$b : Addr \to N_{256}, t \in N_{256}, alive : \mathbb{B}, log : E^k$$

$$f_i(\sigma, v, s, t', \vec{p}) \mapsto \sigma'$$

$$\Phi := \{f_1, ..., f_k\}$$

$$\delta(\sigma, \sigma') := \exists f_i \in \Phi, v, s, t', \vec{p} \,.\, \sigma' = f_i(\sigma, v, s, t', \vec{p})$$

[Shi19]

Построение спецификации

1. Глобальные инварианты на переменных состояния CK

$$balance[owner] > 0$$

2. Отсутствие тупиковых состояний
3. Предикаты EPR FOL на достижимых состояниях

$$\forall s \in State, u_1, u_2 \in Address.$$
$$WP([\![transfer(s, u_1, u_2, v)]\!], True) \wedge$$
$$[s' = transfer(s, u_1, u_2, v)] \implies$$
$$balance(s', u_2) = balance(s, u_2) + v$$

4. Темпоральные формулы на языке событий

$$\bigcup_{u \in Users} Add(u) \cdot (\neg Add(u))^* \cdot Add(u) \cdot [?]^\omega$$

Глобальные инварианты

$$Inv(s) := balance(s, owner) > 0$$

$$\forall s_0 \in State.Initial(s_0) \implies Inv(s_0)$$

$$\forall s, s' \in State.Inv(s) \wedge \delta(s, s') \implies Inv(s')$$

Слабейшее предусловие

$WP([\![f(S, \vec{u})]\!], Post)$ – слабейшее предусловие Дейкстры

$$WP([\![f(S, \vec{u})]\!], Post) \iff [S' = f(S, \vec{u})] \wedge Post(S')$$

$$WP(\langle c_1, ..., c_n \rangle, Pred) = WP(\langle c_1, ..., c_{n-1} \rangle, WP(c_n, Pred))$$

[Dij76]

Слабейшее предусловие

```
contract WP {
  function f(uint x, uint y) external returns (uint) {
    uint res = 0;
    require (x > 100);
    res = x / y;
    return res;
  }
}
```

WP(⟦ function f(x,y) ⟧, f > 2) =
WP(require (x > 100) , res = x / y, return res, f > 2) =
WP(require (x > 100), res = x / y, WP(return res, f > 2)) =
WP(require (x > 100), res = x / y, res > 2) =
WP(require (x > 100), WP(res = x / y, res > 2)) =
WP(require (x > 100), $(x/y > 2) \wedge (y > 0)$) =
$(x > 100) \wedge (x/y > 2) \wedge (y > 0)$

Отсутствие тупиковых состояний

$$\forall s, s' \in State.Initial(s) \wedge \delta^*(s, s') \implies \exists s'', \delta(s', s'')$$

$\delta(s, s')$ — отношение шага

или проверить инвариант

$$enabled(s) := \bigvee_i WP(f_i(s), True)$$

Предикаты EPR FOL на достижимых состояниях

Если выполнилась функция $transfer(u_1, u_2, v)$, то баланс пользователя u_2 увеличился на величину v.

$$\forall s, s' \in State, u_1, u_2 \in Address.$$
$$WP([\![transfer(s, u_1, u_2, v)]\!], True) \wedge [s' = transfer(s, u_1, u_2, v)] \implies$$
$$balance(s', u_2) = balance(s, u_2) + v$$

Темпоральные формулы на языке событий

```
contract ST {
  event A();
  event B();
  event C();
  event D();

  function f1() { emit A(); }
  function f2() { emit B(); emit D(); }
  function f3() { emit C(); }
}
```

$$f1, f2, f3, ... \dashrightarrow A, B, D, C, ...$$

$$f2, f1, f3, ... \dashrightarrow B, D, A, C, ...$$

Требование: Событие C никогда не появится между двумя подряд идущими событиями B и D

$$Spec = \overline{(\neg B)^* \cdot B \cdot (\neg C)^* \cdot C \cdot (\neg B | D)^* \cdot D \cdot [?]^\omega}$$

Темпоральные формулы на языке событий

Нельзя зарегистрировать один и тот же адрес на разные идентификаторы uid.

$$DoubleUidReg(i) := \neg Registered(?, i)^* \cdot Registered(?, i) \cdot \neg Registered(?, i)^* \cdot Registered(?, i) \cdot [?]^\omega$$

$$NeverDoubleUidReg := \overline{\bigcup_{i \in Addr} DoubleUidReg(i)}$$

Отношение включенности подстроки

$$\sigma = \langle \sigma_0, ..., \sigma_k \rangle$$

$$log(\sigma) = log(\sigma_0) +\!\!+ ... +\!\!+ log(\sigma_k)$$

$$\forall \sigma \in \Sigma . substr(log(\sigma), Spec)$$

$$substr(\emptyset, a^* \cdot c)$$

$$substr(\mathsf{a}, a^* \cdot c)$$

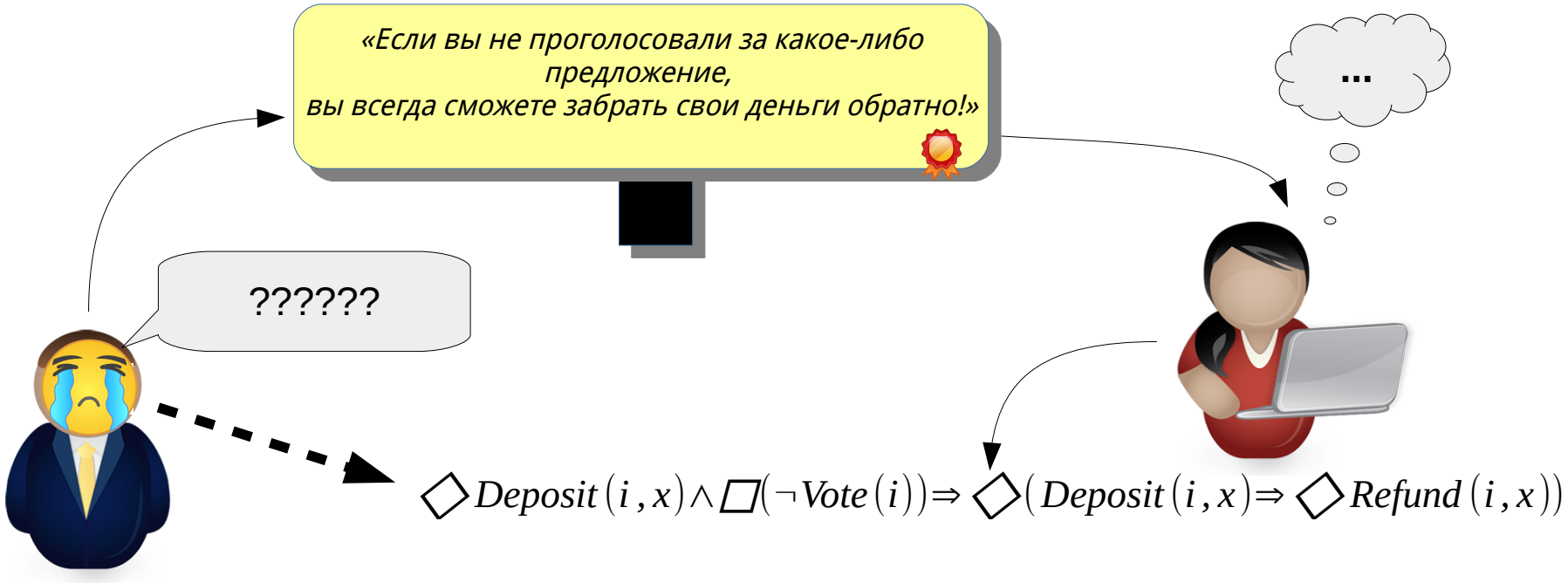
$$substr(\mathsf{c}, a^* \cdot c)$$

$$substr(\mathsf{aac}, a^* \cdot c)$$

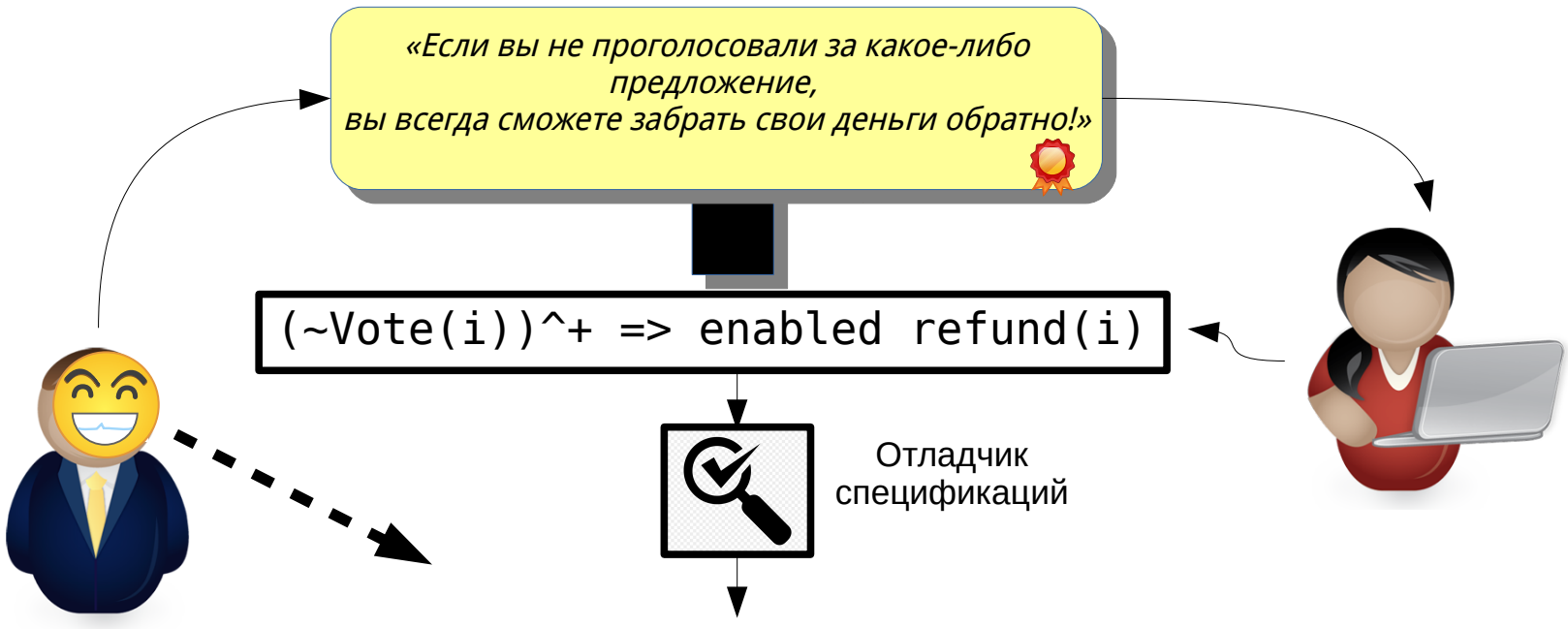
$$\neg substr(\mathsf{acc}, a^* \cdot c)$$

В $Spec$ разрешаются только регулярные выражения, допускающие трансляцию в FOL. При проверке свойства, $substr$ транслируется в ERP-FOL формулу.

Отладка спецификаций



Отладка спецификаций



Построение модели

Построение минимально необходимой модели $Model_i$ для проверки свойства Req_i.

$$IR = sol\_to\_ir(linear(Impl))$$

$$specVars(IR, x > 0 \wedge y > 0) = \{x, y\}$$

$$dep(IR, \{x, y\}) = \{x, y, v_0\}$$

$$dep(IR, \{x, y, v_0\}) = \{x, y, v_0, v_3\}$$

$$dep(IR, \{x, y, v_0, v_3\}) = \{x, y, v_0, v_3\} - \text{ неподвижная точка}$$

$$Model_i = \pi(IR, \{x, y, v_0, v_3\})$$

[BCRZ99]

Устранение симметрии

```
contract BuyTokens {
  mapping (address => uint) balances;
  // ... skipped ...
  function buyTokens(uint tokens) external payable {
    uint tokens = bonus + msg.value / tokenPrice;
    balances[msg.sender] += tokens;
  }
}
```

$$
\begin{aligned}
Req :=& \forall S, S', sender, v. \\
& S' = buyTokens(S, sender, v) \implies \\
& balances(S', sender) = balances(S, sender) + bonus(S) \\
& + v \ \textbf{div} \ tokenPrice
\end{aligned}
$$

Контракт BuyTokens - многопользовательский, но свойство касается одного пользователя и нарушается всегда у пользователя, поэтому его можно рассматривать как однопользовательский.

Верификация модели

```
Vars = {σ_[0..k−1] : Σ, v_[0..k−1] : N_256,  s_[0..k−1] : Addr, t_[0..k−1] : N_256,
        p_[0..k−1] : Π}
i = 0
while (i < k) do {
  if (SAT(I(σ_0) ∧ path(σ_[0..i]) ∧ ¬P(σ_[0..i]), Vars)) {
     print σ_[0..i]
     return false
  }
  i = i + 1
}
return true
```

Listing 1: Проверка предиката на трассе длины k

Существующие разработки - model-checking

- **SMTChecker** ([MOA+20]) - встроенное в компилятор solc средство проверки достижимости оператора $assert$; делает попытку синтеза инвариантов циклов через CHC-движок Z3.
- **VeriSol** ([WLC+18]) - model-checker на базе Boogie/Coral (Z3). Спецификации задаются через $assert$ выражения в коде + глобальные инварианты.
- **VerX** ([PDT+20]) - инструмент для проверки safety-выражений, записанных на safety-фрагменте LTL, синтез инвариантов через CEGAR; нет в публичном доступе.

```
property only_owner_changes_owner {
    always((prev(SampleToken._owner) !=
                 SampleToken._owner)
            ==> (msg.sender == prev(SampleToken._owner)))
                ;
}
```

Существующие разработки - дедуктивные подходы

- **Scilla** ([SNJ+19])- язык программирования смарт-контрактов платформы Zilliqa, напоминающий Gallina (Coq) для упрощения верификации + формальная модель языка
- **InnoChain** (Иннополис) - DSL на базе CakeML для написания смарт-контрактов, с дальнейшей верификацией в Isabelle/HOL.
- **KEVM** ([HSR+18]) - средство дедуктивной верификации байткода EVM в среде $\mathbb{K}$-Framework

Вычислительные сертификаты

Проблема:

- Вычислительная задача верификации модели - затратный по времени и ресурсам процесс
- Как убедить внешнего заказчика, что верификация действительно проводилась, в полном объеме?

Решение:
На помощь приходят *вычислительные сертификаты*.

В статье [SK20] предложен подход на основе линейного доверенного вычислителя.

Источники I


Armin Biere, Edmund Clarke, Richard Raimi, and Yunshan Zhu, *Verifying safety properties of a powerpc- microprocessor using symbolic model checking without bdds*, International Conference on Computer Aided Verification, Springer, 1999, pp. 60–71.


Edsger Wybe Dijkstra, *A discipline of programming*, vol. 613924118, prentice-hall Englewood Cliffs, 1976.


Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, et al., *Kevm: A complete formal semantics of the ethereum virtual machine*, 2018 IEEE 31st Computer Security Foundations Symposium (CSF), IEEE, 2018, pp. 204–217.

Источники II


Matteo Marescotti, Rodrigo Otoni, Leonardo Alt, Patrick Eugster, Antti EJ Hyvärinen, and Natasha Sharygina, *Accurate smart contract verification through direct modelling*, International Symposium on Leveraging Applications of Formal Methods, Springer, 2020, pp. 178–194.


Anton Permenev, Dimitar Dimitrov, Petar Tsankov, Dana Drachsler-Cohen, and Martin Vechev, *Verx: Safety verification of smart contracts*, 2020 IEEE Symposium on Security and Privacy, SP, 2020, pp. 18–20.


Evgeniy Shishkin, *Debugging smart contract's business logic using symbolic model checking*, Programming and Computer Software **45** (2019), no. 8, 590–599.


Evgeny Shishkin and Evgeny Kislitsyn, *Safecomp: Protocol for certifying cloud computations integrity*, arXiv preprint arXiv:2005.10786 (2020).

Источники III

Ilya Sergey, Vaivaswatha Nagaraj, Jacob Johannsen, Amrit Kumar, Anton Trunov, and Ken Chan Guan Hao, *Safer smart contract programming with scilla*, Proceedings of the ACM on Programming Languages **3** (2019), no. OOPSLA, 1–30.

Yuepeng Wang, Shuvendu K Lahiri, Shuo Chen, Rong Pan, Isil Dillig, Cody Born, and Immad Naseer, *Formal specification and verification of smart contracts for azure blockchain*, arXiv preprint arXiv:1812.08829 (2018).