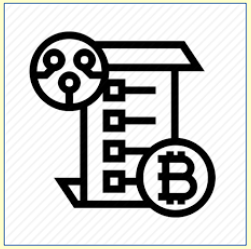
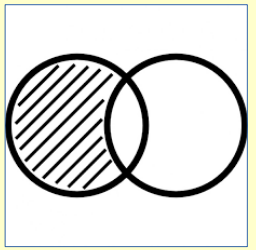
РусКрипто 2018

О Построении Среды для Конструирования Гарантированно Надёжных Смарт-Контрактов




Евгений Шишкин
ИнфоТеКС
2018

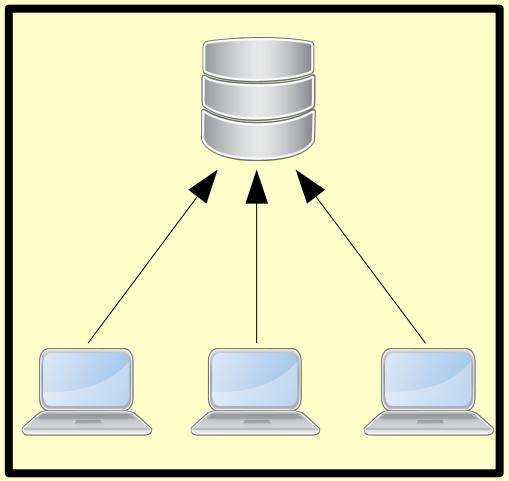



конференция РусКрипто

infotecs®

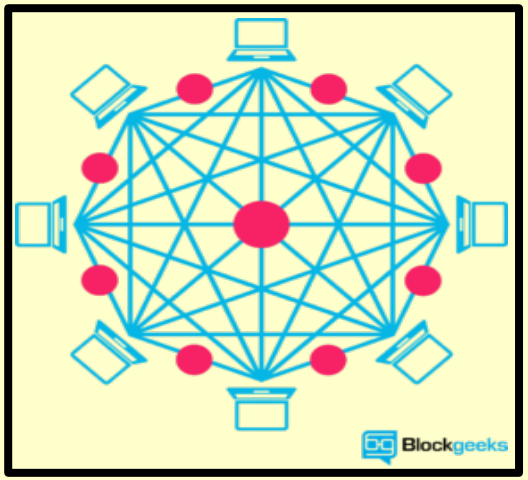
Что такое Блокчейн?

Клиент-Сервер



- Сервер “владеет” данными
- Сервер решает кому дать доступ
- Сервер отвечает за надежное хранение
- Клиенты пользуются “сервисами” сервера

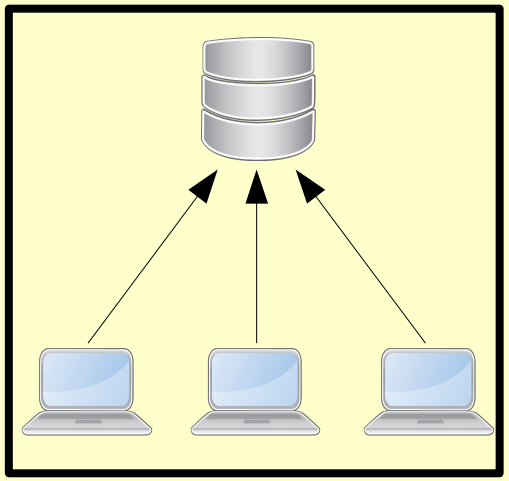
Блокчейн



- Избыточность данных
- Отсутствует администратор
- Данные невозможно подделать
- История проведенных операций
- **Мотивация к корректному выполнению протокола**

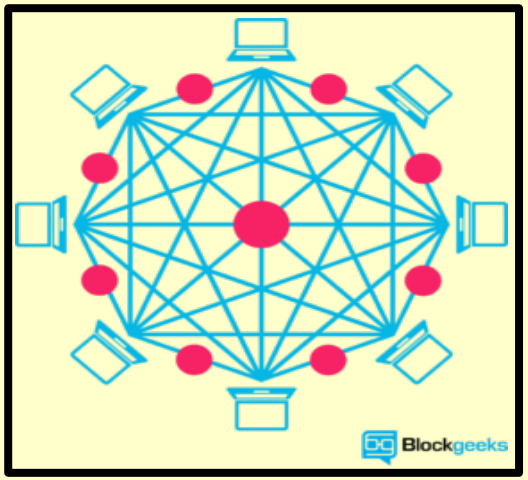
Что такое Смарт-Контракт?

Клиент-Сервер



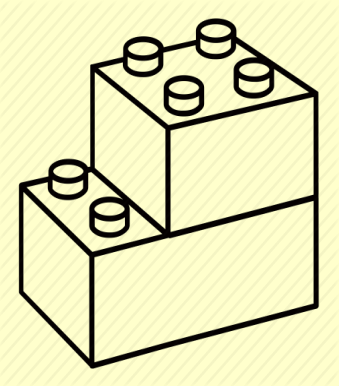
- Сервер предоставляет клиентам сервисы, логика их работы не видна
- Взаимная аутентификация
- Пользовательские данные хранятся на сервере
- Вычисления происходят в основном на сервере

Блокчейн

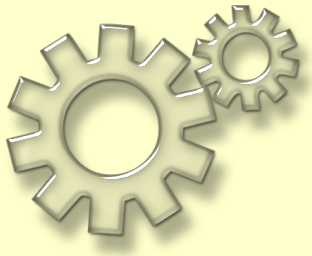


- Загружаемые пользователями программы; логика программ доступна для изучения
- Псевдоанонимные пользователи
- История вызовов и состояние программы сохраняется на каждом узле и общедоступно
- Вычисления дублируются и перепроверяются на каждом узле системы

Бизнес-ценность Технологии

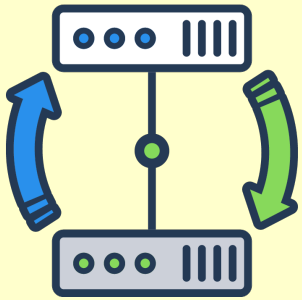


Неизменяемая бизнес-логика между сторонами без изначального доверия



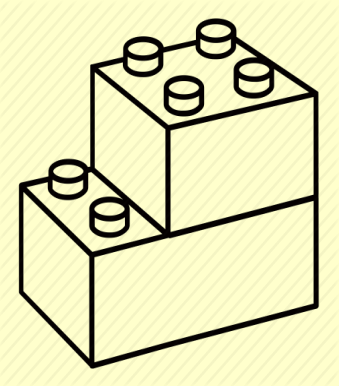
Прозрачность операций

Возможность взаиморасчетов внутри системы



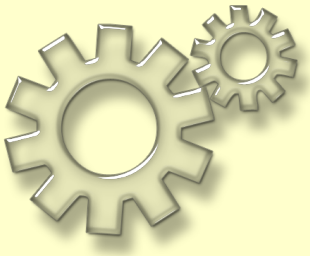
Высокая доступность
Отказоустойчивость

Обратная сторона



Неизменяемая бизнес-логика между сторонами без изначального доверия

Трудно либо невозможно исправить ошибку в логике

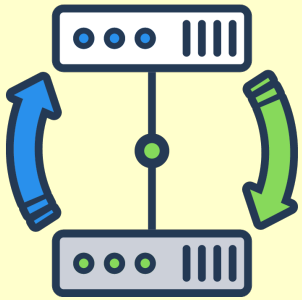


Прозрачность операций

Приватность? Анонимность?

Возможность взаиморасчетов внутри системы

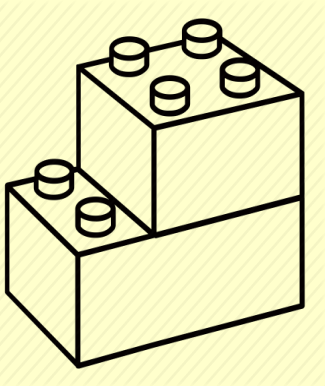
Кража криптовалюты



Высокая доступность
Отказоустойчивость

Нет возможности уйти на “технический перерыв”

Нас интересует вот эта проблема



Неизменяемая бизнес-логика между сторонами без изначального доверия

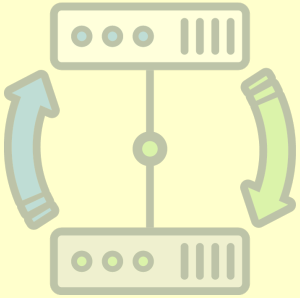
Трудно либо невозможно исправить ошибку в контракте!

Прозрачность операций

Приватность? Анонимность?

Возможность взаиморасчетов внутри системы

Кража криптовалюты



Высокая доступность
Отказоустойчивость

Нет возможности уйти на “технический перерыв”

Выявление дефектов

Категория	Назначение	Образцы
Linters	Поиск отклонений в исходном коде от лучших практик по безопасному программированию; читаемость и т д	Solint
Static Analysis	Поиск операционных ошибок в программе	Oyente, SmartCheck
Test Coverage	Фреймворк для построения и управления тестами	Solidity-coverage, Truffle
Formal Verification Tools	Строгий полу-автоматический аудит по формальной спецификации	Solidity-to-Why3

Автоматизация

Глубина выявления дефектов

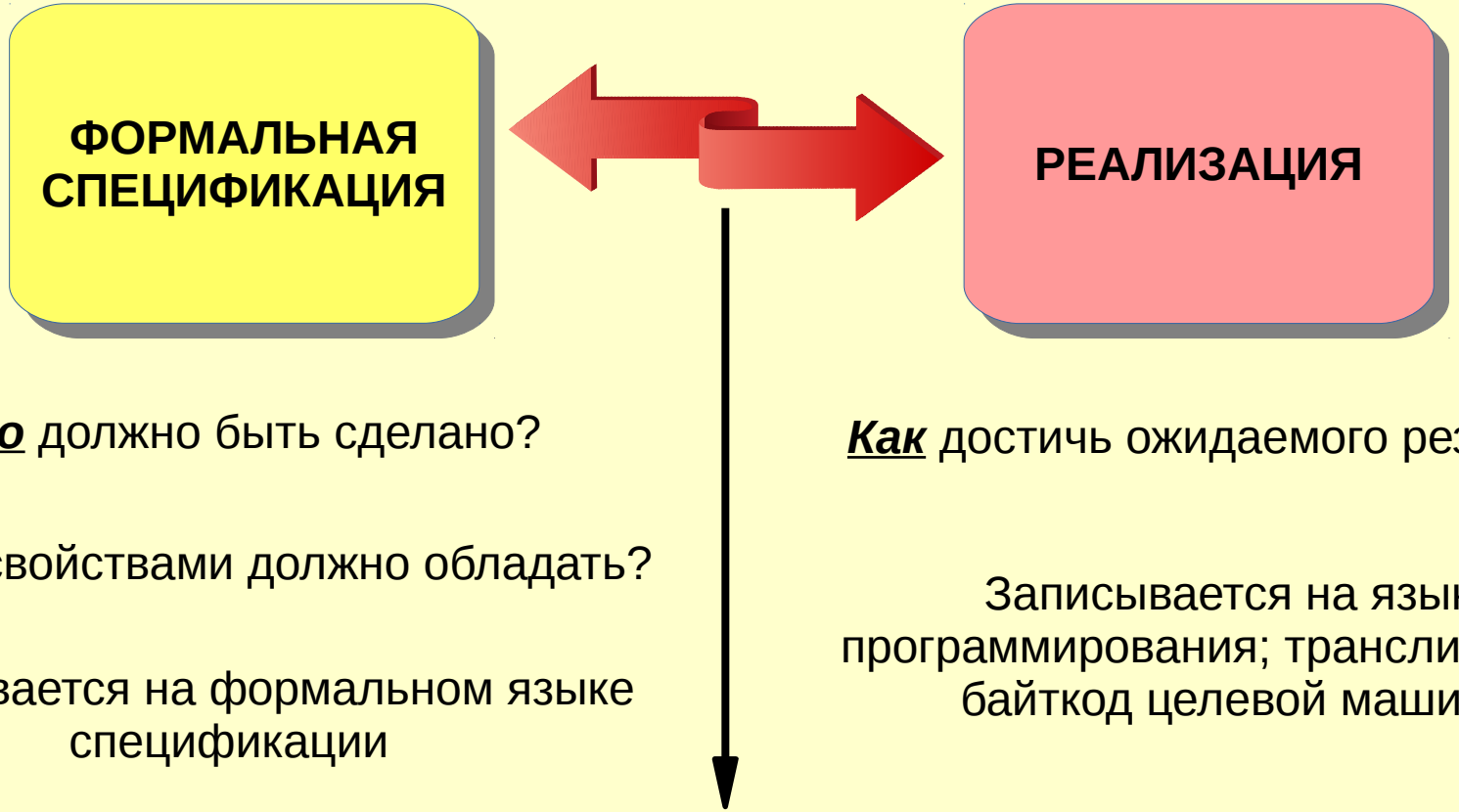
Крайняя точка спектра

Категория	Назначение	Образцы
Linters	Поиск отклонений в исходном коде от лучших практик по безопасному программированию; читаемость и т д	Solint
Static Analysis	Поиск операционных ошибок в программе	Oyente, SmartCheck
Test Coverage	Фреймворк для построения и управления тестами	Solidity-coverage, Truffle
Formal Verification Tools	Строгий полу-автоматический аудит по формальной спецификации	Solidity-to-Why3

Автоматизация

Глубина выявления дефектов

Формальная верификация



Что должно быть сделано?

Какими свойствами должно обладать?

Записывается на формальном языке спецификации

Как достичь ожидаемого результата

Записывается на языке программирования; транслируется в байткод целевой машины

Систематический, теоретически обоснованный метод соотнесения реализации и формальной спецификации

Пример спецификации

Спецификация алгоритма в логике Хоара

$$\{P\}C\{Q\}$$

$$\underline{\textbf{\textit{precond}}\ 1}: beg, end \in \mathbb{N} \wedge end \geq beg$$

$$\underline{\textbf{\textit{postcond}}\ 1}: \forall i \in \mathbb{N}, beg \leq i < end \Rightarrow a'[i] \leq a'[i+1]$$

$$\textbf{\textit{postcond}}\ 2:$$
$$\forall i \in \mathbb{N}, \exists j \in \mathbb{N}, beg \leq i, j \leq end \Rightarrow a'[i] = a[j]$$

$$\textbf{\textit{postcond}}\ 3:$$
$$\forall i \in \mathbb{N}, \exists j \in \mathbb{N}, beg \leq i, j \leq end \Rightarrow a[i] = a'[j]$$

Спецификация функции сортировки

$$\{precond\ 1\}$$
$$quicksort(a, beg, end)$$
$$\{postcond\ 1 \wedge postcond\ 2 \wedge postcond\ 3\}$$

Реализация алгоритма на языке Си

```
int partition ( int a [ ], int beg, int end ) {
  int left , right , loc , flag = 0, pivot ;
  loc = left = beg;
  right = end;
  pivot = a [ loc ] ;
  while ( flag == 0 )
  {
    while( (pivot <= a [ right ] )&&( loc != right ) )
          right - - ;
    if( loc == right )  flag = 1;
    else {
     a  [ loc ] = a [ right ] ;
     left = loc + 1 ;
     loc = right;
    }
    while ( (pivot  >= a [ left ] ) && ( loc != left ) )
         left++;
    if( loc == left ) flag = 1;
    else {
      a [ loc ] = a  [ left ] ;
      right = loc - 1;
      loc = left;
    }
  }
  a [ loc ] = pivot;
  return loc;
}
```

```
void quick_sort(int a[ ] , int beg , int end ) {
    int loc;
    if ( beg < end ) {
       loc = partition( a , beg , end );
       quick_sort ( a , beg , loc – 1 );
       quick_sort ( a , loc + 1 , end );
    }
}
void print_array (int a [ ],int n)  {
  int i;
  for ( i = 0 ; I < n ; i++ ) printf( "%5d" ,a [ i ] ) ;
}
int main () {
  int count , num[ 50 ] , i ;
  printf ("How many elements to sort : ");
  scanf ("%d", &count );
  printf ("\n Enter the elements : \n\n");
  for( i = 0; i < count; i++ )  {
     printf ("num [%d ]  : " , i );
     scanf( "%d", &num[ i ] );
  }
  printf (" \n Array Before Sorting : \n\n\n" );
  print_array ( num , count ) ;
  quick_sort ( num ,0 , count-1) ;
  printf ( "\n\n\n Array  After Sorting : \n\n\n" );
  print_array ( num , count );
}
```

Доменные логики

Название Логики	Предметная область	Комментарий
Логика Хоара	Последовательные императивные программы с массивами и циклами	Хоар, 1969 г.
Сепарационная Логика	Последовательные императивные программы с массивами, циклами, кучей и указателями	Рейнольдс, О'Хирн, 2002 г.
Линейная темпоральная логика	Распределенные асинхронные системы	Пнуэли, 1977, Лампорт, 1994
Линейная логика	Управление ресурсами	Гирард, 1987
Логика рациональных агентов	Мультиагентные системы	Вулдридж, 1995

Трансформационные vs Реагирующие системы

	Трансформационные системы	Реагирующие системы
Примеры	Сжатие информации, Шифрование, Вычислительные функции	Коммуникационные протоколы, облачные сервисы, ядро ОС
Цель	Получение выходного значения как функции исходного значения	Взаимодействие с окружающим миром посредством реакций на возникающие события
Характер вычислений	Строго завершающиеся, результат доступен по завершению	Бесконечный цикл реакций на возникающие события, завершаются только в некоторых случаях
Семантика	Функция. Выход программы является функцией входных данных	Бесконечная цепочка состояний. Последовательность состояний и реакций системы на внешние события

* Таблица составлена по мотивам аналогичной таблицы из книги Ю.Г.Карпова “ModelChecking – верификация параллельных и распределенных программных систем”

Типичная реагирующая система

Smart Contract

Ethereum Account Type (Just like User Account)

Address — *0x16E0022b17B...*

Balance — *0 Ether*

Code

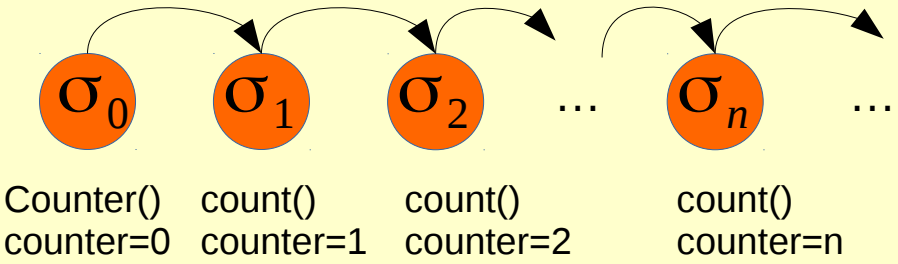
State

```
contract Counter {
    uint counter;

    function Counter() public {
        counter = 0;
    }
    function count() public {
        counter = counter + 1;
    }
}
```

Линейная темпоральная логика

```
contract Counter {
  uint counter;
  function Counter() public {
      counter = 0;
  }
  function count() public {
      uint c = counter + 1;
      assert (c > counter);
      counter = c;
  }
}
```



Список состояний системы $\sigma = \sigma_0\sigma_1\sigma_2\ldots\sigma_n\ldots$ мы называем ***трассой***.
Множество всех возможных трасс системы образуют ***поведение*** системы.

“Состояние” контракта = набор значений переменных контракта

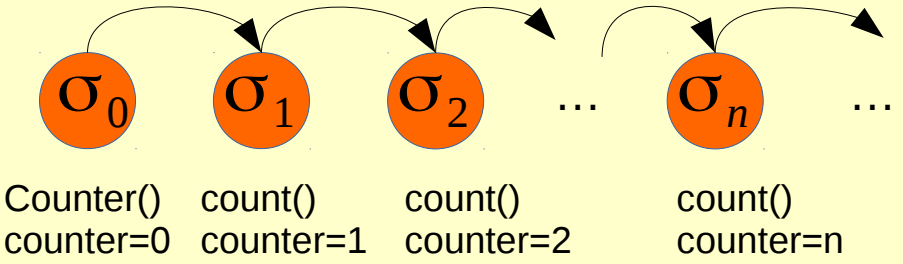
$$\sigma \models P \Leftrightarrow P\,\sigma_0 \qquad \sigma \models \circ P \Leftrightarrow P\,\sigma_1$$

$$\sigma \models \Diamond P \Leftrightarrow \exists i \geq 0, P\,\sigma_i \qquad \sigma \models \Box P \Leftrightarrow \forall i \geq 0, P\,\sigma_i$$

$$\sigma \models P\,\mathbf{U}\,Q \Leftrightarrow \exists k \geq 0, \sigma^k \models Q \wedge \forall\, j, 0 \leq j < k \Rightarrow \sigma^j \models P$$

Примеры LTL-спецификаций

```
contract Counter {
  uint counter;
  function Counter() public {
      counter = 0;
  }
  function count() public {
      uint c = counter + 1;
      assert (c > counter);
      counter = c;
  }
}
```



Так ли это?!

$$\Diamond(counter = 5)$$

$$\Box(counter > 0)$$

$$\Diamond(counter = 5 \Rightarrow \Box(counter \geq 5))$$

В приведенном варианте темпоральной логики нет возможности выразить отношение между настоящим и прошлым, но такой вариант логики возможен.

Верификация

```
contract Counter {
  uint counter;
  function Counter() public {
      counter = 0;
  }
  function count() public {
      uint c = counter + 1;
      assert (c > counter);
      counter = c;
  }
}
```

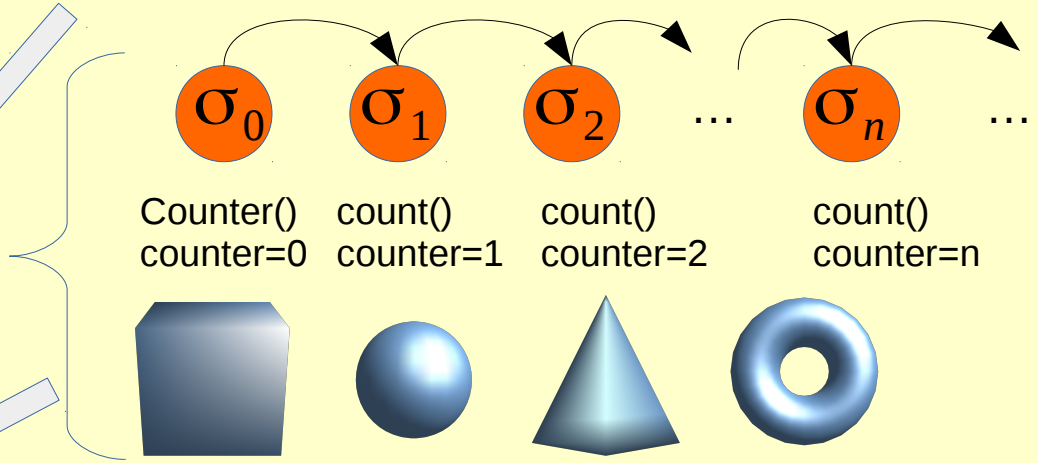
Как убедить себя в выполнении данного свойства?

$$\Diamond(counter = 5 \Rightarrow \Box(counter \geq 5))$$

Общая архитектура среды

```
contract Counter {
  uint counter;
  function Counter() public {
      counter = 0;
  }
  function count() public {
      uint c = counter + 1;
      assert (c > counter);
      counter = c;
  }
}
```

$$\Diamond\left(counter = 5 \Rightarrow \Box\left(counter \geq 5\right)\right)$$



Solidity-to-Gallina

Кодирование LTL свойств в логике Coq

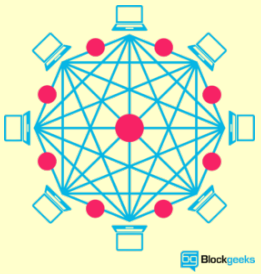
Кодирование нужных абстракций в Coq

Coq Proof Assistant (INRIA, France)

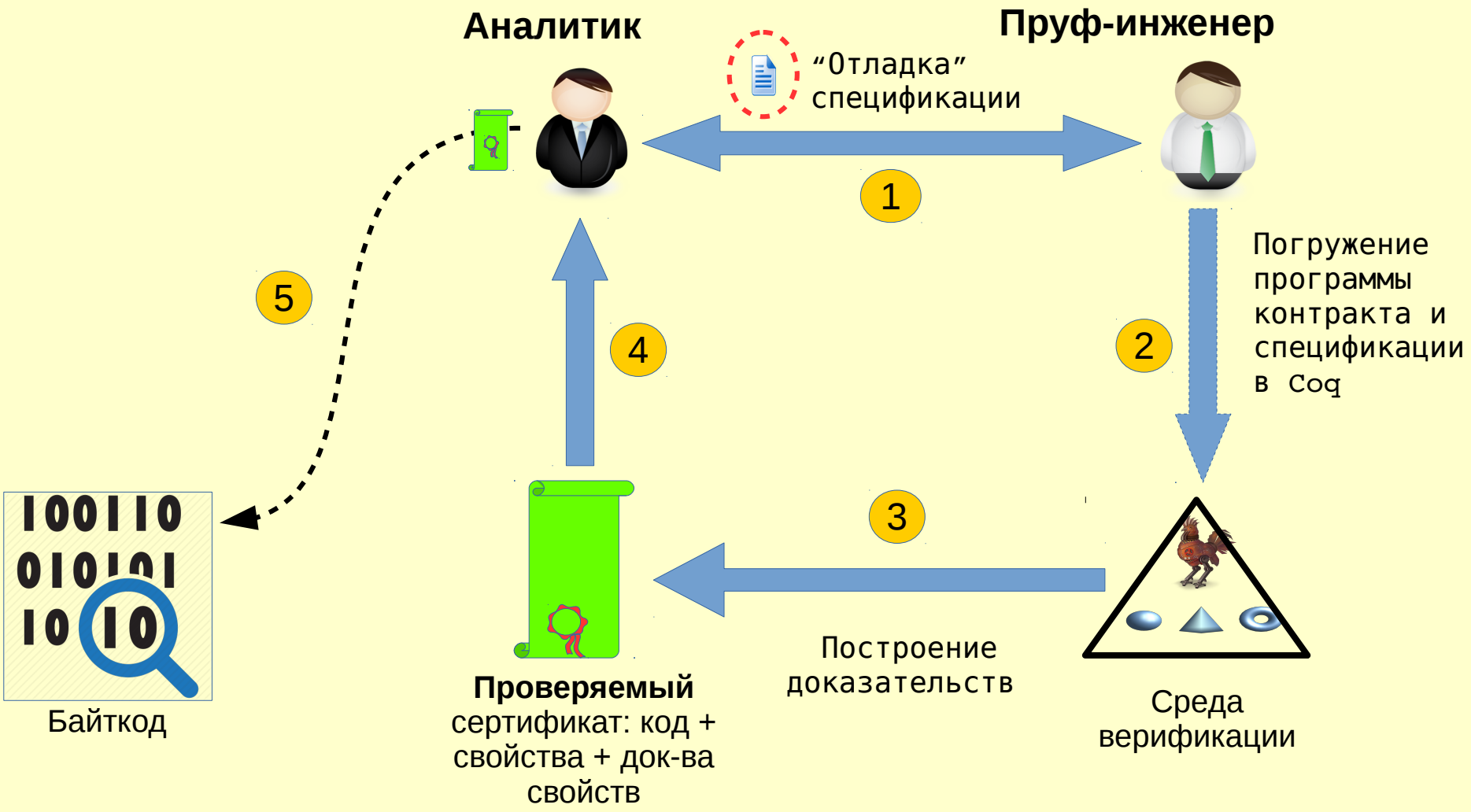
Экспорт верифицированного кода в EVM-байткод

Байткод

Публикация в блокчейне



Сценарий использования



Компоненты Среды

Название	Назначение	Комментарий
Теория поведения смарт-контрактов	Позволяет описать систему, состоящую из одного смарт-контракта, в который поступают вызовы;	Набор индуктивных и ко-индуктивных типов внутри логики Coq
Теория LTL	позволяет записать желаемые свойства контакта в логике LTL	Библиотека Coq Отсутствует модальность прошлого; возможно добавится
SpecChecker	Генератор достижимых состояйний контракта с проверкой выполнимости заданных свойств	Прототип; может проверять только свойства типа “всегда P”
Транслятор	Перевод программы, записанной на языке Gallina, в нечто, что может быть скомпилировано в EVM	Компонент отсутствует; перевод осуществляется ручным переводом конструкций

Сертификат представляется в виде набора файлов с доказательствами.

Итоги

- Обозначена необходимость тщательной проверки смарт-контрактов перед публикацией в блокчейне
- Описан один из возможных подходов – формальная верификация программы контракта относительно спецификации в виде утверждений на языке линейной темпоральной логики
- Описано концептуальное устройство среды для такой верификации
- Описан сценарий использования среды в промышленном контексте
- Исследовательская работа находится на начальном этапе; мы рассчитываем на коллаборацию с практиками аудита для валидации подхода и получения обратной связи.

Дискуссия

Евгений Шишкин
Ведущий исследователь ЦНИПР
ИнфоТеКС
Email: evgeniy.shishkin@gmail.com