Язык Ride - простой функциональный язык смарт-сонтрактов.

Михаил Потанин.
Scala-разработчик Waves Platform.
mpotanin@wavesplatform.com

Язык Ride это язык смарт-контрактов в блокчейне Waves.

Язык Ride это язык смарт-контрактов в блокчейне Waves.

Язык очень простой

Простой синтаксис.

```
let alicePubKey  =
        base58'5Azf.....VdEpMM'
let bobPubKey    =
        base58'2KwU.....wi2VDF'
let cooperPubKey =
        base58'GbrU.....mgt5cD'

func check(proof: Int, key: ByteVector) =
  if sigVerify(tx.bodyBytes, tx.proofs[proof], key)
  then 1
  else 0

#check whoever provided the valid proof
let aliceSigned  = check(0, alicePubKey)
let bobSigned    = check(1, bobPubKey)
let cooperSigned = check(2, cooperPubKey)

#sum up every valid proof to get at least 2
aliceSigned + bobSigned + cooperSigned >= 2
```

Что такое смарт-контракты?

Что такое смарт-контракты?

Договор, оформленный в виде кода.

Концепцию предложил Ник Сабо в 1996 году для удешевления составления договоров для микроплатежей.

Основное применение смарт-контрактов в настоящее время - блокчейн.

Основное применение смарт-контрактов в настоящее время - блокчейн.

Но ничто не мешает применять их более широко:

В банковской сфере можно формально описать условия выплат.

Основное применение смарт-контрактов в настоящее время - блокчейн.

Но ничто не мешает применять их более широко:

В банковской сфере можно формально описать условия выплат.

В многопользовательских играх с развитой экономикой.

К какой информации имеет доступ смарт-контракт и какие действия он может инициировать?

К какой информации имеет доступ смарт-контракт и какие действия он может инициировать?

В игре он может иметь доступ ко всем элементам игровой вселенной, которые разрешены правилами игры.

К какой информации имеет доступ смарт-контракт и какие действия он может инициировать?

В игре он может иметь доступ ко всем элементам игровой вселенной, которые разрешены правилами игры.

Для банков - это данные из различных реестров и введенные оператором.

К какой информации имеет доступ смарт-контракт и какие действия он может инициировать?

В игре он может иметь доступ ко всем элементам игровой вселенной, которые разрешены правилами игры.

Для банков - это данные из различных реестров и введенные оператором.

В блокчейне это только данные самого блокчейна или помещенные в блокчейн доверенным третьей стороной (оракулом).

Кто пишет смарт-контракты.

Кто пишет смарт-контракты.

Сейчас это обычные программисты.

Кто пишет смарт-контракты.

Сейчас это обычные программисты.

В идеале, это юристы и владельцы бизнесов.

Цена ошибки.

Цена ошибки.

В результате программисткой ошибки в смарт-контракте пользователь может потерять деньги.

Цена ошибки.

В результате программисткой ошибки в смарт-контракте пользователь может потерять деньги.

К счету пользователя может получить доступ злоумышленник.

Цена ошибки.

В результате программисткой ошибки в смарт-контракте пользователь может потерять деньги.

К счету пользователя может получить доступ злоумышленник.

Счет может оказаться заблокирован.

Как уменьшить вероятность ошибок?

Как уменьшить вероятность ошибок?

Классический ответ - тестирование.

Как уменьшить вероятность ошибок?

Классический ответ - тестирование.

Тестировать смарт-контракты необходимо, но это не панацея.

Как уменьшить вероятность ошибок?

Что может предложить язык.

Как уменьшить вероятность ошибок?

Что может предложить язык.

Иммутабельность и отсутствие побочных эффектов.

Как уменьшить вероятность ошибок?

Что может предложить язык.

Иммутабельность и отсутствие побочных эффектов.

Статическая типизация.

Как уменьшить вероятность ошибок?

Что может предложить язык.

Иммутабельность и отсутствие побочных эффектов.

Статическая типизация: в Ride реализованы union-типы.

Union-типы.

```
func incOrSize(v: Int | String) = match v {
   case a: Int    => a+1
   case b: String => size(b)
}
```

Union-типы.

```
match getInteger(tx.sender, "p14") {
    case _: Unit => 0
    case v: Int  => v
}
```

Как уменьшить вероятность ошибок?

Что может предложить язык.

Иммутабельность и отсутствие побочных эффектов.

Статическая типизация: в Ride реализованы union-типы.

Формальная верификация.

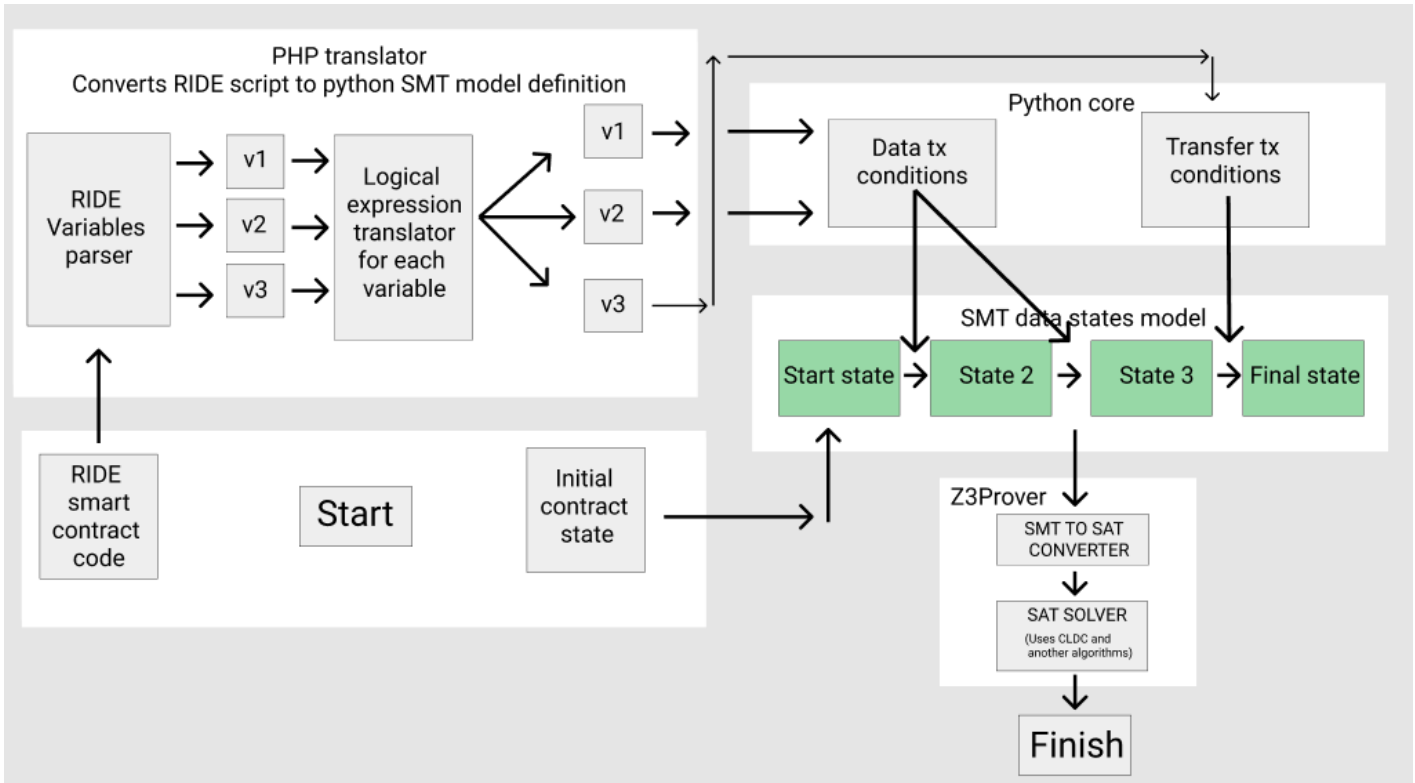
Как уменьшить вероятность ошибок?

Формальная верификация.

Благодоря простоте появились внешние проекты

Формальная верификация.

https://habr.com/en/post/450016/



Модель выполнения Ride.

Модель выполнения Ride.

Валидация транзакций.

Модель выполнения Ride.

Валидация транзакций.

Контракт это функция из транзакции и текущего состояния блокчейна в булевский тип

```
contract : Transaction -> Blockchain -> Boolean
```

Ошибки времени исполнения.

Ошибки времени исполнения.

Ошибки приводят к завершению контракта. Тразнакция в этом случае отклоняется.

Ошибки времени исполнения.

Ошибки приводят к завершению контракта. Тразнакция в этом случае отклоняется.

Возможности обработки ошибок не предусмотрено.

Ошибки в подвыражениях и ленивость.

```
let x = 1/0
true
```

Ошибки в подвыражениях и ленивость.

```
let x = 1/0
true
```

вычисляется в true, потому что к x нет обращений.

Неполнота по Тьюрингу.

Неполнота по Тьюрингу.

В Ride запрещена рекурсия.

Неполнота по Тьюрингу.

В Ride запрещена рекурсия.

В Ride нет функций высших порядков.

Последовательность операций.

Последовательность операций.

Поведение смарт-контракта может зависеть от сохраненных ранее данных.

Последовательность операций.

Поведение смарт-контракта может зависеть от сохраненных ранее данных.

Сохранение данных - специальный вид транзакции, который тоже проверяется смарт-контрактом.

Последовательность операций.

Поведение смарт-контракта может зависеть от сохраненных ранее данных.

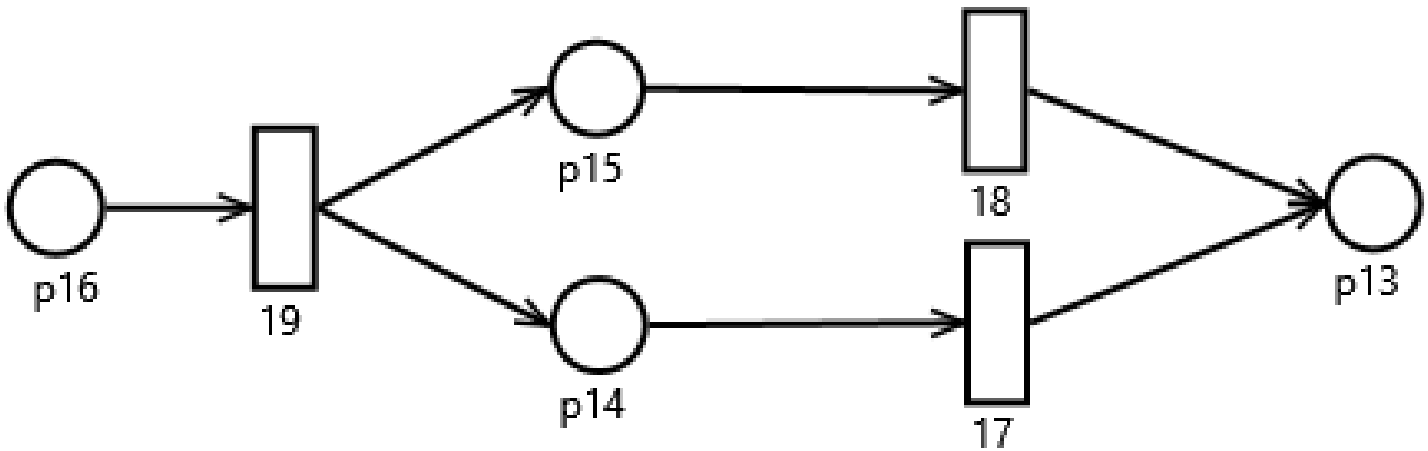
Сохранение данных - специальный вид транзакции, который тоже проверяется смарт-контрактом.

Смарт-контракт может валидировать каждый шаг бизнес-процесса.

Бизнес-процесс часто можно описать в простой формальной модели - конечного автомата или сети Петри.

Бизнес-процесс часто можно описать в простой формальной модели - конечного автомата или сети Петри.

Которые легко реализуются смарт-контрактом.


p16
19
p15
p14
18
17
p13

```
match tx {
   case tx: DataTransaction =>
     let action = getInteger(tx.data, "action")
        if action == 17 && size(tx.data) == 3
      then
        let p14 = match getInteger(tx.sender, "p14") { case _:Unit => 0 case v: Int => v }
        let p13 = match getInteger(tx.sender, "p13") { case _:Unit => 0 case v: Int => v }
           p14 >= 1 && p14 + -1 == extract(getInteger(tx.data, "p14"))
                    && p13 + 1 == extract(getInteger(tx.data, "p13"))
      else if action == 18 && size(tx.data) == 3
      then
        let p15 = match getInteger(tx.sender, "p15") { case _:Unit => 0 case v: Int => v }
        let p13 = match getInteger(tx.sender, "p13") { case _:Unit => 0 case v: Int => v }
           p15 >= 1 && p15 + -1 == extract(getInteger(tx.data, "p15"))
                    && p13 + 1 == extract(getInteger(tx.data, "p13"))
      else if action == 19 && size(tx.data) == 4
      then
        let p14 = match getInteger(tx.sender, "p14") { case _:Unit => 0 case v: Int => v }
        let p15 = match getInteger(tx.sender, "p15") { case _:Unit => 0 case v: Int => v }
        let p16 = match getInteger(tx.sender, "p16") { case _:Unit => 0 case v: Int => v }
           p16 >= 1 && p14 + 1 == extract(getInteger(tx.data, "p14"))
                    && p15 + 1 == extract(getInteger(tx.data, "p15"))
                    && p16 + -1 == extract(getInteger(tx.data, "p16"))
      else throw("Action is not implemented")
    case tx =>
        true
}
```

Один вызов смарт-контракта может соответствовать шагу более сложной формальной системы.

Один вызов смарт-контракта может соответствовать шагу более сложной формальной системы.

В том числе тьюринг-полной.

Один вызов смарт-контракта может соответствовать шагу более сложной формальной системы.

В том числе тьюринг-полной.

Подробно описано на примере Ergo в статье Александра Чепурного, Василия Харина и Дмитрия Мешкова.

"Self-Reproducing Coins as Universal Turing Machine"
https://arxiv.org/pdf/1806.10116.pdf

Разделяя приложение на производящий вычисления клиент и валидирующий смарт-контракт можно реализовать любой DApp.

Разделяя приложение на производящий вычисления клиент и валидирующий смарт-контракт можно реализовать любой DApp.

Но сложно.

Разделяя приложение на производящий вычисления клиент и валидирующий смарт-контракт можно реализовать любой DApp.

Но сложно.

Основная сложность - в обеспечении атомарности нескольких транзакций.

Разделяя приложение на производящий вычисления клиент и валидирующий смарт-контракт можно реализовать любой DApp.

Но сложно.

Основная сложность - в обеспечении атомарности нескольких транзакций.

Решение проблемы - Ride for DApps!

Другая модель исполнения смартконтакта.

Другая модель исполнения смартконтакта.

Пользователь только инициирует запуск кода контракта, а контракт решает, что должно произойти.

```
@Callable(i)
func deposit() = {
   let pmt = extract(i.payment)
   if (isDefined(pmt.assetId)) then throw("can hodl waves only at the moment")
   else {
        let currentKey = toBase58String(i.caller.bytes)
        let currentAmount = match getInteger(this, currentKey) {
            case a:Int => a
            case _ => 0
        }
        let newAmount = currentAmount + pmt.amount
        WriteSet([DataEntry(currentKey, newAmount)])
   }
}
```

```
@Callable(i)
func withdraw(amount: Int) = {
   let currentKey = toBase58String(i.caller.bytes)
   let currentAmount = match getInteger(this, currentKey) {
       case a:Int => a
       case _ => 0
   }
   let newAmount = currentAmount - amount
   if (amount < 0)
           then throw("Can't withdraw negative amount")
   else if (newAmount < 0)
           then throw("Not enough balance")
           else ScriptResult(
                   WriteSet([DataEntry(currentKey, newAmount)]),
                   TransferSet([ScriptTransfer(i.caller, amount, waves)]))
}
```

Действия, порождаемые контрактом, выполняются атомарно.

Действия, порождаемые контрактом, выполняются атомарно.

Если одно из них не может быть выполнено, вся транзакция исполнения скрипта отменяется.

Спасибо за внимание!

Михаил Потанин.
mpotanin@wavesplatform.com
https://github.com/potan
https://habr.com/ru/users/potan/
https://t.me/potan/