Хорошо протестировать нетестируемое и не сойти с ума

Денис Буздало́в

Heisenbug 2024 Autumn
Санкт-Петербург
17 октября 2024

Что такое *хорошее тестирование*?

Что такое *хорошее тестирование*?

- Находит баги?

Что такое *хорошее тестирование*?

- Находит баги?
- Покрывает код?

Что такое *хорошее тестирование*?

- Находит баги?
- Покрывает код?
- Покрывает функциональность/требования?

Что такое *хорошее тестирование*?

- Находит баги?
- Покрывает код?
- Покрывает функциональность/требования?
- Повышает уверенность?

Что такое *хорошее тестирование*?

- Находит баги?
- Покрывает код?
- Покрывает функциональность/требования?
- Повышает уверенность?
- Много тестов?

Что такое *хорошее тестирование*?

- Находит баги?
- Покрывает код?
- Покрывает функциональность/требования?
- Повышает уверенность?
- Много тестов?
- Быстро работает?

Что такое *хорошее тестирование*?

- Находит баги?
- Покрывает код?
- Покрывает функциональность/требования?
- Повышает уверенность?
- Много тестов?
- Быстро работает?
- Легко разрабатывается?

Что такое *хорошее тестирование*?

- Находит баги?
- Покрывает код?
- Покрывает функциональность/требования?
- Повышает уверенность?
- Много тестов?
- Быстро работает?
- Легко разрабатывается?
- Хотя бы присутствует?

Авиационная байка

Фото: The Aviation Herald

Несколько фактов о самолётах

Несколько фактов о самолётах

- второй круг

Несколько фактов о самолётах

- второй круг
- даже если касание шасси

Несколько фактов о самолётах

- второй круг
- даже если касание шасси
- торможение шасси и двигателем (реверс)

Несколько фактов о самолётах

- второй круг
- даже если касание шасси
- торможение шасси и двигателем (реверс)
- реверс в полёте опасен, это предусмотрено

Несколько фактов о самолётах

- второй круг
- даже если касание шасси
- торможение шасси и двигателем (реверс)
- реверс в полёте опасен, это предусмотрено
- 11 февраля 1978, Pacific Airline 314

Несколько фактов о самолётах

- второй круг
- даже если касание шасси
- торможение шасси и двигателем (реверс)
- реверс в полёте опасен, это предусмотрено
- 11 февраля 1978, Pacific Airline 314
- сертификационное требование:

$$\text{посадка} \longrightarrow \text{реверс} \longrightarrow \text{выключение} \longrightarrow \text{прямая тяга} \longrightarrow \text{взлёт}$$

Несколько фактов о самолётах

- второй круг
- даже если касание шасси
- торможение шасси и двигателем (реверс)
- реверс в полёте опасен, это предусмотрено
- 11 февраля 1978, Pacific Airline 314
- сертификационное требование:

$$\text{посадка} \longrightarrow \text{реверс} \longrightarrow \text{выключение} \longrightarrow \text{прямая тяга} \longrightarrow \text{взлёт}$$

- Airbus 320, двигатель CFM56, логика:

$$\left.\begin{array}{r}\text{команда ухода на второй круг}\\ \text{стойка под двигателем ”на земле”}\end{array}\right\} \longrightarrow \text{свернуть реверс}$$

Фото:Report 2022-150, Serious incident to CS-TNV (Airbus A320-214) in Copenhagen/Kastrup (EKCH) on 8-4-2022

8 апреля 2022, TAP Air Portugal 754

Фото:Report 2022-150, Serious incident to CS-TNV (Airbus A320-214) in Copenhagen/Kastrup (EKCH) on 8-4-2022

8 апреля 2022, TAP Air Portugal 754

8 апреля 2022, TAP Air Portugal 754

Фото:Report 2022-150, Serious incident to CS-TNV (Airbus A320-214) in Copenhagen/Kastrup (EKCH) on 8-4-2022

8 апреля 2022, TAP Air Portugal 754

- уход на второй круг — команда, не состояние

8 апреля 2022, TAP Air Portugal 754

- уход на второй круг — команда, не состояние
- cross-wind bounce

8 апреля 2022, TAP Air Portugal 754

- уход на второй круг — команда, не состояние
- cross-wind bounce
- пока команда давалась (0,18 с.), правое шасси было на земле, а левое в воздухе

8 апреля 2022, TAP Air Portugal 754

- уход на второй круг — команда, не состояние
- cross-wind bounce
- пока команда давалась (0,18 с.), правое шасси было на земле, а левое в воздухе
- левый двигатель не свернул реверс

8 апреля 2022, TAP Air Portugal 754

- уход на второй круг — команда, не состояние
- cross-wind bounce
- пока команда давалась (0,18 с.), правое шасси было на земле, а левое в воздухе
- левый двигатель не свернул реверс
- redundancy: была подсистема, которая сделала auto idle

Хорошее тестирование?

- софт был корректно сертифицирован

Хорошее тестирование?

- софт был корректно сертифицирован
- 95 миллионов посадок

Хорошее тестирование?

- софт был корректно сертифицирован
- 95 миллионов посадок
- оценки исправления софта — к 2025

Хорошее тестирование?

- софт был корректно сертифицирован
- 95 миллионов посадок
- оценки исправления софта — к 2025
- изменения в процедуре сертификации

Хорошее тестирование?

- софт был корректно сертифицирован
- 95 миллионов посадок
- оценки исправления софта — к 2025
- изменения в процедуре сертификации
- такого не найдено в других конфигурациях

Хорошее тестирование

Я утверждаю: одно из свойств *хорошего метода хорошего тестирования* — ставить тестируемую систему в такие ситуации (и находить такие баги), о которых автор тестов даже не думал

Property-based testing

Property-based testing

- Тестирование функции/системы на *произвольном* входе

Property-based testing

- Тестирование функции/системы на *произвольном* входе
- Оценка, а не предугадывание результата

Property-based testing

- Тестирование функции/системы на *произвольном* входе
- Оценка, а не предугадывание результата
- Рандомизированная генерация входных значений

Property-based testing

- Тестирование функции/системы на *произвольном* входе
- Оценка, а не предугадывание результата
- Рандомизированная генерация входных значений
- Минимизация контрпримеров (shrinking)

Property-based testing

- Тестирование функции/системы на *произвольном* входе
- Оценка, а не предугадывание результата
- Рандомизированная генерация входных значений
- Минимизация контрпримеров (shrinking)
- (Иногда) автоматическая деривация генераторов и минимизаторов

Property-based testing

- Тестирование функции/системы на *произвольном* входе
- Оценка, а не предугадывание результата
- Рандомизированная генерация входных значений
- Минимизация контрпримеров (shrinking)
- (Иногда) автоматическая деривация генераторов и минимизаторов
- Десятки библиотек под множество языков

  Haskell, Erlang, Scala, Python, Coq, Idris, C#, C++, Clojure, D, Elixir, Elm, F#, Go, Java, JavaScript, Julia, Kotlin, Nim, OCaml, Prolog, Racket, Ruby, Rust, Swift, TypeScript, ...

Свойство?

```
insertOk : Property
insertOk = property $ do
  insert 2 [1, 3, 5] === [1, 2, 3, 5]
```

Свойство?

```
insertOk : Property
insertOk = property $ do
  insert ?x ?xs === ?result
```

Свойство?

```
insertOk : Property
insertOk = property $ do
  x  <- forAll arbitraryNat
  insert x ?xs === ?result
```

Свойство?

```
insertOk : Property
insertOk = property $ do
  x  <- forAll arbitraryNat
  xs <- forAll sortedNatList
  insert x xs === ?result
```

Свойство!

```
insertOk : Property
insertOk = property $ do
  x  <- forAll arbitraryNat
  xs <- forAll sortedNatList
  assert $ sorted $ insert x xs
```

Свойство!

```
insertOk : Property
insertOk = property $ do
  x  ← forAll arbitraryNat
  xs ← forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

Свойство!

```
insertOk : Property
insertOk = property $ do
  x  <- forAll arbitraryNat
  xs <- forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

```
── sorted list insertion ──
  ✓ insertOk passed 100 tests.
```

Свойство!

```
insertOk : Property
insertOk = property $ do
  x  ← forAll arbitraryNat
  xs ← forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

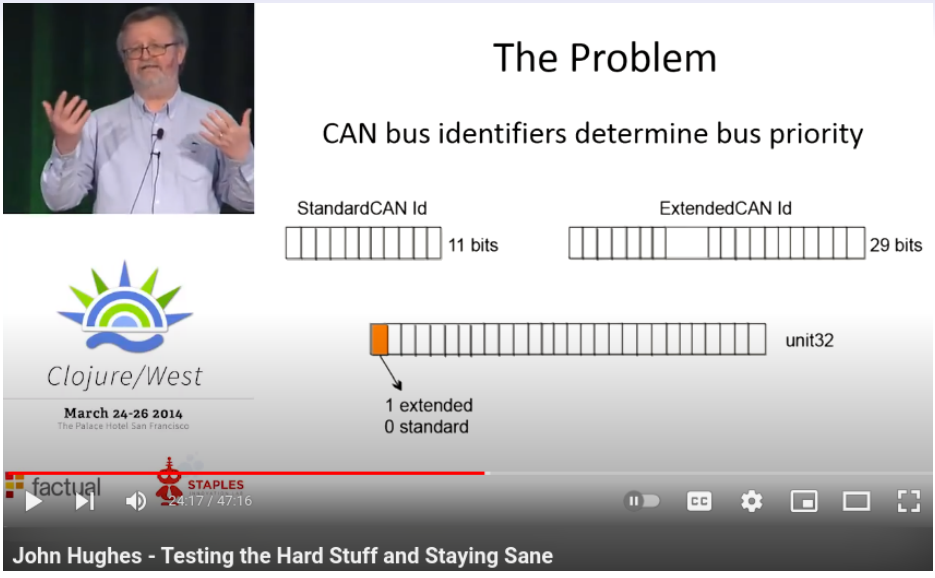
А что, если `insert x xs = x :: xs`?

```
insertOk : Property
insertOk = property $ do
  x  <- forAll arbitraryNat
  xs <- forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

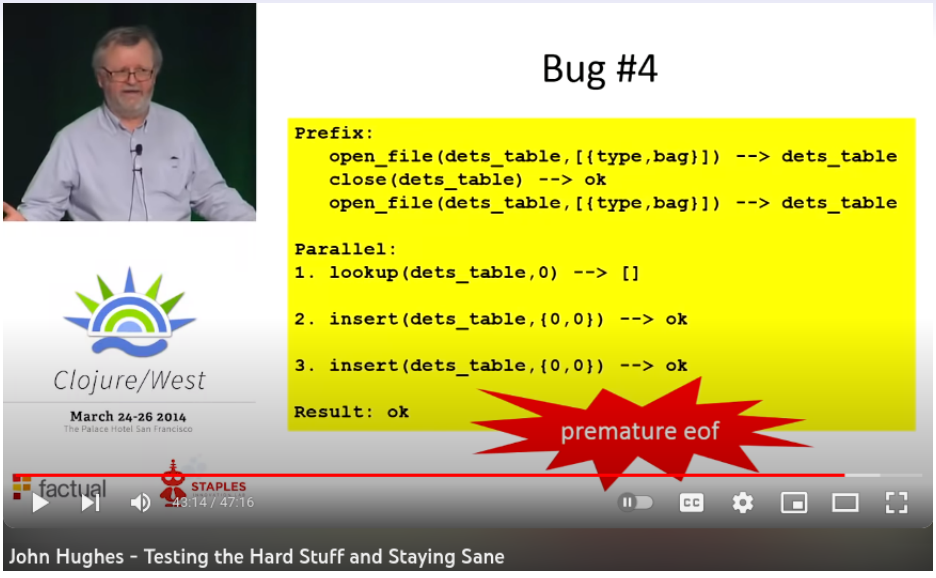
А что, если `insert x xs = x :: xs`?

```
insertOk : Property
insertOk = property $ do
  x  <- forAll arbitraryNat
  xs <- forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

```
── sorted list insertion ──
  ✗ insertOk failed after 14 tests.

    forAll 0 =
      1

    forAll 1 =
      [0]
```



[1] https://www.youtube.com/watch?v=zi0rHwfiX1Q



[1] https://www.youtube.com/watch?v=zi0rHwfiX1Q

Хорошо применённый property-based testing

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах
- ✔ высокоуровневая спецификация может находить низкоуровневые проблемы

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах
- ✔ высокоуровневая спецификация может находить низкоуровневые проблемы
- ✔ может найти практически ненаходимое ручным тестированием

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах
- ✔ высокоуровневая спецификация может находить низкоуровневые проблемы
- ✔ может найти практически ненаходимое ручным тестированием
- ✔ находит то, о чём не подозревали, *хороший* метод

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах
- ✔ высокоуровневая спецификация может находить низкоуровневые проблемы
- ✔ может найти практически ненаходимое ручным тестированием
- ✔ находит то, о чём не подозревали, *хороший* метод
- ✔ сложность тестирования растёт медленнее сложности SUT

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах
- ✔ высокоуровневая спецификация может находить низкоуровневые проблемы
- ✔ может найти практически ненаходимое ручным тестированием
- ✔ находит то, о чём не подозревали, *хороший* метод
- ✔ сложность тестирования растёт медленнее сложности SUT
- ✘ надо уметь выбирать подходящие свойства

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах
- ✔ высокоуровневая спецификация может находить низкоуровневые проблемы
- ✔ может найти практически ненаходимое ручным тестированием
- ✔ находит то, о чём не подозревали, *хороший* метод
- ✔ сложность тестирования растёт медленнее сложности SUT
- ✘ надо уметь выбирать подходящие свойства
- ✘ reimplementation trap

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах
- ✔ высокоуровневая спецификация может находить низкоуровневые проблемы
- ✔ может найти практически ненаходимое ручным тестированием
- ✔ находит то, о чём не подозревали, *хороший* метод
- ✔ сложность тестирования растёт медленнее сложности SUT
- ✘ надо уметь выбирать подходящие свойства
- ✘ reimplementation trap
- ✘ формализация требований, нужен опыт и mindsetting

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах
- ✔ высокоуровневая спецификация может находить низкоуровневые проблемы
- ✔ может найти практически ненаходимое ручным тестированием
- ✔ находит то, о чём не подозревали, *хороший* метод
- ✔ сложность тестирования растёт медленнее сложности SUT
- ✘ надо уметь выбирать подходящие свойства
- ✘ reimplementation trap
- ✘ формализация требований, нужен опыт и mindsetting
  - ✔ инварианты, модели, метаморфное тестирование, автоматы

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах
- ✔ высокоуровневая спецификация может находить низкоуровневые проблемы
- ✔ может найти практически ненаходимое ручным тестированием
- ✔ находит то, о чём не подозревали, *хороший* метод
- ✔ сложность тестирования растёт медленнее сложности SUT
- ✘ надо уметь выбирать подходящие свойства
- ✘ reimplementation trap
- ✘ формализация требований, нужен опыт и mindsetting
  - ✔ инварианты, модели, метаморфное тестирование, автоматы
- ✘ написание генераторов, корректность, полнота, распределения

Хорошо применённый property-based testing

- ✔ одна спецификация находит ошибки в разных местах
- ✔ высокоуровневая спецификация может находить низкоуровневые проблемы
- ✔ может найти практически ненаходимое ручным тестированием
- ✔ находит то, о чём не подозревали, *хороший* метод
- ✔ сложность тестирования растёт медленнее сложности SUT
- ✘ надо уметь выбирать подходящие свойства
- ✘ reimplementation trap
- ✘ формализация требований, нужен опыт и mindsetting
  - ✔ инварианты, модели, метаморфное тестирование, автоматы
- ✘ написание генераторов, корректность, полнота, распределения
  - ✔ деривация

Деривация?

```
insertOk : Property
insertOk = property $ do
  x  <- forAll arbitraryNat
  xs <- forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

Деривация?

```
arbitraryNat : Gen Nat
```

```
sortedNatList : Gen (List Nat)
```

```
insertOk : Property
insertOk = property $ do
  x  <- forAll arbitraryNat
  xs <- forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

Деривация?

```
arbitraryNat : Gen Nat
arbitraryNat = deriveGen
```

```
sortedNatList : Gen (List Nat)
```

```
insertOk : Property
insertOk = property $ do
  x  <- forAll arbitraryNat
  xs <- forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

Деривация?

```
arbitraryNat : Gen Nat
arbitraryNat = deriveGen

arbitraryNatList : Gen (List Nat)


sortedNatList : Gen (List Nat)



insertOk : Property
insertOk = property $ do
  x  <- forAll arbitraryNat
  xs <- forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

Деривация?

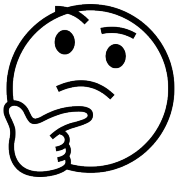
```
arbitraryNat : Gen Nat
arbitraryNat = deriveGen

arbitraryNatList : Gen (List Nat)


sortedNatList : Gen (List Nat)
sortedNatList =
  foldr (\x, res ⇒ x :: map (+x) res) [] <$> arbitraryNatList

insertOk : Property
insertOk = property $ do
  x  ← forAll arbitraryNat
  xs ← forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

Деривация?

```
arbitraryNat : Gen Nat
arbitraryNat = deriveGen

arbitraryNatList : Gen (List Nat)
arbitraryNatList = deriveGen

sortedNatList : Gen (List Nat)
sortedNatList =
  foldr (\x, res ⇒ x :: map (+x) res) [] <$> arbitraryNatList

insertOk : Property
insertOk = property $ do
  x  ← forAll arbitraryNat
  xs ← forAll sortedNatList
  assert $ sorted $ insert x xs
  assert $ x `elem` insert x xs
```

Деривация?

Вот бы можно было выражать свои *намерения* так, чтобы компилятор сам вывел генератор



Списки

```
data NatList : Type where
  Nil  : NatList
  (::) : Nat -> NatList -> NatList
```

Списки

```
data NatList : Type where
  Nil  : NatList
  (::) : Nat → NatList → NatList

actuallySorted : NatList
actuallySorted = 1 :: 2 :: 5 :: 9 :: Nil
```

Списки

```
data NatList : Type where
  Nil  : NatList
  (::) : Nat → NatList → NatList

actuallySorted : NatList
actuallySorted = 1 :: 2 :: 5 :: 9 :: Nil

unsorted : NatList
unsorted = 1 :: 5 :: 2 :: 9 :: 1 :: Nil
```

Списки сортированные?

```
data NatList : Type where
  Nil  : NatList
  (::) : Nat -> NatList -> NatList
```

Списки сортированные?

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : Nat -> SortedNatList -> SortedNatList
```

Списки сортированные?

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : Nat → SortedNatList → SortedNatList
         -- голова должна быть не больше головы хвоста
         -- (если есть)
```

Списки сортированные?

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : (x : Nat) → (xs : SortedNatList) → SortedNatList
         -- `x` не больше головы `xs` (если есть)
```

Списки сортированные?

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : (x : Nat) -> (xs : SortedNatList) -> SortedNatList
         -- `x` не больше головы `xs` (если есть)

data LTEHead : Nat -> SortedNatList -> Type where
  NoHead   : LTEHead n Nil
  SomeHead : So (n < x) -> LTEHead n (x::xs)
```

Списки сортированные!

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : (x : Nat) -> (xs : SortedNatList) ->
         LTEHead x xs => SortedNatList

data LTEHead : Nat -> SortedNatList -> Type where
  NoHead   : LTEHead n Nil
  SomeHead : So (n < x) -> LTEHead n $ (x::xs) @{_}
```

Списки сортированные!

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : (x : Nat) -> (xs : SortedNatList) ->
         LTEHead x xs => SortedNatList

data LTEHead : Nat -> SortedNatList -> Type where
  NoHead   : LTEHead n Nil
  SomeHead : So (n < x) -> LTEHead n $ (x::xs) @{_}

actuallySorted : SortedNatList
actuallySorted = 1 :: 2 :: 5 :: 9 :: Nil
```

Списки сортированные!

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : (x : Nat) -> (xs : SortedNatList) ->
         LTEHead x xs => SortedNatList

data LTEHead : Nat -> SortedNatList -> Type where
  NoHead   : LTEHead n Nil
  SomeHead : So (n < x) -> LTEHead n $ (x::xs) @{_}

actuallySorted : SortedNatList
actuallySorted = 1 :: 2 :: 5 :: 9 :: Nil

failing "Can't find an implementation for LTEHead"
  unsorted : SortedNatList
  unsorted = 1 :: 5 :: 2 :: 9 :: 1 :: Nil
```

Списки сортированные!

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : (x : Nat) -> (xs : SortedNatList) ->
         LTEHead x xs => SortedNatList

data LTEHead : Nat -> SortedNatList -> Type where
  NoHead   : LTEHead n Nil
  SomeHead : So (n < x) -> LTEHead n $ (x::xs) @{_}

arbitrarySortedNatList : Gen SortedNatList
```

Списки сортированные!

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : (x : Nat) -> (xs : SortedNatList) ->
         LTEHead x xs => SortedNatList

data LTEHead : Nat -> SortedNatList -> Type where
  NoHead   : LTEHead n Nil
  SomeHead : So (n < x) -> LTEHead n $ (x::xs) @{_}

arbitrarySortedNatList : Fuel -> Gen MaybeEmpty SortedNatList
```

Списки сортированные!

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : (x : Nat) -> (xs : SortedNatList) ->
         LTEHead x xs => SortedNatList

data LTEHead : Nat -> SortedNatList -> Type where
  NoHead   : LTEHead n Nil
  SomeHead : So (n < x) -> LTEHead n $ (x::xs) @{_}

arbitrarySortedNatList : Fuel -> Gen MaybeEmpty SortedNatList
arbitrarySortedNatList = deriveGen
```

Списки сортированные!

```
data SortedNatList : Type where
  Nil  : SortedNatList
  (::) : (x : Nat) -> (xs : SortedNatList) ->
         LTEHead x xs => SortedNatList

data LTEHead : Nat -> SortedNatList -> Type where
  NoHead   : LTEHead n Nil
  SomeHead : So (n < x) -> LTEHead n $ (x::xs) @{_}

arbitrarySortedNatList : Fuel -> Gen MaybeEmpty SortedNatList
arbitrarySortedNatList = deriveGen
```

Списки сортированные!

```
     ____    _         _             ___
    /  _/___/ /____(_)____   |__ \
    / // __  / ___/ / ___/   __/ /     Version 0.7.0-0e83d6c7c
  _/ // /_/ / /  / (__  )   / __/      https://www.idris-lang.org
 /___/\__,_/_/  /_/____/   /____/      Type :? for help

Welcome to Idris 2.  Enjoy yourself!
Main> :exec pick (arbitrarySortedNatList $ limit 100) >>= printLn . map toList
Just [56, 57, 74, 76, 89, 92, 95]
Main> :exec pick (arbitrarySortedNatList $ limit 100) >>= printLn . map toList
Just [7, 8, 53, 67, 91, 94, 96, 97]
Main> :exec pick (arbitrarySortedNatList $ limit 100) >>= printLn . map toList
Just [57, 94, 99]
Main>
```

Пример из жизни

Пример из жизни

- Диалект Typescript

Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор

Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка

Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Свойства

Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Свойства
  - семантически корректные программы должны компилироваться

Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Свойства
  - семантически корректные программы должны компилироваться
  - среди них завершающиеся программы должны интерпретироваться без неожиданных падений и зацикливаний

Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Свойства
  - семантически корректные программы должны компилироваться
  - среди них завершающиеся программы должны интерпретироваться без неожиданных падений и зацикливаний
  - все варианты запуска должны выдавать одинаковый результат

Как это можно специфицировать

```
data Stmts : (functions  : List (Name, FunSig)) ->
             (varsBefore : List (Name, Type)) ->
             (varsAfter  : List (Name, Type)) -> Type where

  (.)  : (ty : Type) -> (n : Name) ->
         Stmts funs vars ((n, ty)::vars)

  (#=) : (n : Name) -> (0 lk : n `IsIn` vars) =>
         (v : Expr funs vars (found lk)) ->
         Stmts funs vars vars

  If   : (cond : Expr funs vars Bool) ->
         Stmts funs vars vThen -> Stmts funs vars vElse ->
         Stmts funs vars vars

  (>>) : Stmts funs preV midV  -> Stmts funs midV postV ->
         Stmts funs preV postV
```

Как это можно специфицировать

```
record FunSig where
  constructor (⟹)
  From : List Type
  To   : Type

data Expr : List (Name, FunSig) → List (Name, Type) →
            Type → Type where

  C : (x : ty) → Expr funs vars ty

  V : (n : Name) → (0 lk : n `IsIn` vars) ⇒
      Expr funs vars (found lk)

  F : (n : Name) → (0 lk : n `IsIn` funs) ⇒
      All (Expr funs vars) (found lk).From →
      Expr funs vars (found lk).To
```

Семантически корректные программы

```
StdF : List (Name, FunSig)
StdF = [ ("+" , [Int, Int] ⟹ Int)
       , ("<" , [Int, Int] ⟹ Bool)
       , ("++", [Int] ⟹ Int)
       , ("||", [Bool, Bool] ⟹ Bool) ]
```

Семантически корректные программы

```
StdF : List (Name, FunSig)
StdF = [ ("+" , [Int, Int] ⟹ Int)
       , ("<" , [Int, Int] ⟹ Bool)
       , ("++", [Int] ⟹ Int)
       , ("||", [Bool, Bool] ⟹ Bool) ]
```

```
program : Stmts StdF [] ?
program = do
  Int. "x"
  "x" #= C 5
  Int. "y"; Bool. "res"
  "y" #= F "+" [V "x", C 1]
```

Семантически корректные программы

```
StdF : List (Name, FunSig)
StdF = [ ("+" , [Int, Int] ⟹ Int)
       , ("<" , [Int, Int] ⟹ Bool)
       , ("++", [Int] ⟹ Int)
       , ("||", [Bool, Bool] ⟹ Bool) ]
```

```
program : Stmts StdF [] ?
program = do
  Int. "x"
  "x" #= C 5
  Int. "y"; Bool. "res"
  "y" #= F "+" [V "x", C 1]
  If (F "<" [F "++" [V "x"], V "y"])
     (do "y" #= C 0; "res" #= C False)
     (do Int. "z"; "z" #= F "+" [V "x", V "y"]
         Bool. "b"; "b" #= F "<" [V "x", C 5]
         "res" #= F "||" [V "b", F "<" [V "z", C 6]])
```

Семантически некорректные программы

```
failing "Mismatch between: Int and Bool"
  bad : Stmts StdF [] ?
  bad = do
    Int. "x"; "x" #= C 5
    Bool. "y"; "y" #= F "+" [V "x", C 1]
```

Семантически некорректные программы

```
failing "Mismatch between: Int and Bool"
  bad : Stmts StdF [] ?
  bad = do
    Int. "x"; "x" #= C 5
    Bool. "y"; "y" #= F "+" [V "x", C 1]

failing "Mismatch between: [] and [Int]"
  bad : Stmts StdF [] ?
  bad = do
    Int. "x"; "x" #= C 5
    Int. "y"; "y" #= F "+" [V "x"]
```

Семантически некорректные программы

```
failing "Mismatch between: Int and Bool"
  bad : Stmts StdF [] ?
  bad = do
    Int. "x"; "x" #= C 5
    Bool. "y"; "y" #= F "+" [V "x", C 1]

failing "Mismatch between: [] and [Int]"
  bad : Stmts StdF [] ?
  bad = do
    Int. "x"; "x" #= C 5
    Int. "y"; "y" #= F "+" [V "x"]

failing "Mismatch between: Bool and Int"
  bad : Stmts StdF [] ?
  bad = do
    Int. "x"; "x" #= C 5
    Int. "y"; "y" #= F "+" [C True, V "x"]
```

Применим

Testing...

Применим

```
ASSERTION FAILED: false
IN /.../optimizations/lse.cpp:851: GetEliminationCode
Backtrace [tid=2990599]:
#0 : 0x7f876a1776c0 PrintStack(std::ostream&)
#1 : 0x7f876a177562 debug::AssertionFail(...)
#2 : 0x7f8767185e10 compiler::Lse::GetEliminationCode(...)
#3 : 0x7f8767186a20 compiler::Lse::DeleteInstructions(...)
...
```

Применим

```
ASSERTION FAILED: false
IN /.../optimizations/lse.cpp:851: GetEliminationCode
Backtrace [tid=2990599]:
#0 : 0x7f876a1776c0 PrintStack(std::ostream&)
#1 : 0x7f876a177562 debug::AssertionFail(...)
#2 : 0x7f8767185e10 compiler::Lse::GetEliminationCode(...)
#3 : 0x7f8767186a20 compiler::Lse::DeleteInstructions(...)
...
```

Shrinking...

Применим

```
class C0 {
  x0: boolean
}

function main() : void {
  let x1: C0 = {x0: true}
  while(x1.x0) {
    x1.x0 = x1.x0
    x1.x0 = false
  }
}
```

Применим

Testing...

Применим

```
Wrong input 0 type 'i32' for inst:
    52.ref  NullCheck  v42, v51 -> (v55, v53)
                                        bc: 0x0000005d
ASSERTION FAILED: CheckType(GetInputType(inst, 0), ...)
IN /.../inst_checker_gen.h:694: VisitNullCheck
ERRNO: 29 (Illegal seek)
Backtrace [tid=3853514]:
#0 : 0x7f46fc7b393c PrintStack(std::ostream&)
#1 : 0x7f46fc7b37de debug::AssertionFail(...)
#2 : 0x7f46fe3760ad compiler::InstChecker::VisitNullCheck(...)
#3 : 0x7f46fe38dae5 compiler::InstChecker::VisitGraph()
#4 : 0x7f46fe35e63e compiler::InstChecker::Run(...)
#5 : 0x7f46fe33c1b2 compiler::GraphChecker::Check()
...
```

Применим

```
Wrong input 0 type 'i32' for inst:
    52.ref  NullCheck  v42, v51 → (v55, v53)
                                        bc: 0x0000005d
ASSERTION FAILED: CheckType(GetInputType(inst, 0), ...)
IN /.../inst_checker_gen.h:694: VisitNullCheck
ERRNO: 29 (Illegal seek)
Backtrace [tid=3853514]:
#0 : 0x7f46fc7b393c PrintStack(std::ostream&)
#1 : 0x7f46fc7b37de debug::AssertionFail(...)
#2 : 0x7f46fe3760ad compiler::InstChecker::VisitNullCheck(...)
#3 : 0x7f46fe38dae5 compiler::InstChecker::VisitGraph()
#4 : 0x7f46fe35e63e compiler::InstChecker::Run(...)
#5 : 0x7f46fe33c1b2 compiler::GraphChecker::Check()
...
```

Shrinking...

Применим

```
function main() {
   for(let x2 of [0]) {
     let x3: boolean = false
     for(let x4 of [0]) {
       let x5: int[][] = [[]]
       let fuel1 = 0
     }
   }
   let fuel0 = 0
}
```

Применим

Testing...

Применим

```
ASSERTION FAILED: block->GetGraph() == GetGraph()
IN /.../optimizer/ir/graph_cloner.h:176: GetClone
Backtrace [tid=2902033]:
#0 : 0x7fe71892b820 PrintStack(std::ostream&)
#1 : 0x7fe71892b6c2 debug::AssertionFail(...)
#2 : 0x7fe71a61ae61 compiler::GraphCloner::GetClone(...)
#3 : 0x7fe71a61162a compiler::GraphCloner::CopyLoop(...)
#4 : 0x7fe71a611839 compiler::GraphCloner::CopyLoop(...)
#5 : 0x7fe71a611173 compiler::GraphCloner::CloneAnalyses(...)
#6 : 0x7fe71a610d1f compiler::GraphCloner::CloneGraph()
#7 : 0x7fe71a5b377c compiler::GraphChecker::GraphChecker(...)
...
```

Применим

```
ASSERTION FAILED: block->GetGraph() == GetGraph()
IN /.../optimizer/ir/graph_cloner.h:176: GetClone
Backtrace [tid=2902033]:
#0 : 0x7fe71892b820 PrintStack(std::ostream&)
#1 : 0x7fe71892b6c2 debug::AssertionFail(...)
#2 : 0x7fe71a61ae61 compiler::GraphCloner::GetClone(...)
#3 : 0x7fe71a61162a compiler::GraphCloner::CopyLoop(...)
#4 : 0x7fe71a611839 compiler::GraphCloner::CopyLoop(...)
#5 : 0x7fe71a611173 compiler::GraphCloner::CloneAnalyses(...)
#6 : 0x7fe71a610d1f compiler::GraphCloner::CloneGraph()
#7 : 0x7fe71a5b377c compiler::GraphChecker::GraphChecker(...)
...
```

Shrinking...

Применим

```
class C0 {
  x0: boolean

  f() : string {
    return ""
  }
}
```

```
function main() : void {
  let x2: C0 = {x0: true}
  let fuel0 = 1
  while(fuel0 > 0) {
    do {
      fuel0--
      do {
        fuel0--
        let s = x2.f()
      } while(true && (fuel0 > 0))
    } while(true && (fuel0 > 0))
  }
}
```

Применим

Testing...

Применим

```
TypeError: Unreachable statement. [<filename>:26:34]
```

Применим

```
TypeError: Unreachable statement. [<filename>:26:34]
```

Shrinking...

Применим

```
TypeError: Unreachable statement. [<filename>:3:30]
```

```
function main() : void {
  let x1: Int = 1
  while(([false, true])[x1]) {
  }
}
```

Применим

- ...и так далее

Применим

- ...и так далее
- Было найдено 9 подобных ошибок
  - В JIT-, AOT-оптимизаторе, тайпчекере, кодогенераторе

Применим

- ...и так далее
- Было найдено 9 подобных ошибок
  - В JIT-, AOT-оптимизаторе, тайпчекере, кодогенераторе
- Ещё 8 во время написания спецификации

Применим

- ...и так далее
- Было найдено 9 подобных ошибок
  - В JIT-, AOT-оптимизаторе, тайпчекере, кодогенераторе
- Ещё 8 во время написания спецификации
- Наша спецификация

Применим

- ...и так далее
- Было найдено 9 подобных ошибок
  - В JIT-, AOT-оптимизаторе, тайпчекере, кодогенераторе
- Ещё 8 во время написания спецификации
- Наша спецификация
  - Подмножество
    - Завершающиеся программы
    - Циклы, ветвления, присваивания, исключения
    - Классы без методов, числа, массивы

Применим

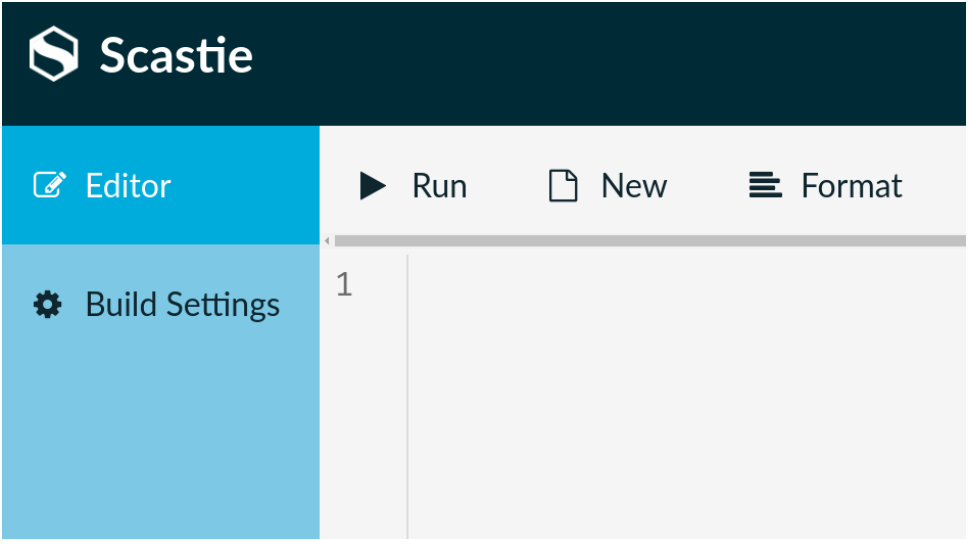
- ...и так далее
- Было найдено 9 подобных ошибок
  - В JIT-, AOT-оптимизаторе, тайпчекере, кодогенераторе
- Ещё 8 во время написания спецификации
- Наша спецификация
  - Подмножество
    - Завершающиеся программы
    - Циклы, ветвления, присваивания, исключения
    - Классы без методов, числа, массивы
  - ~330 строк кода спецификация языка + столько же обвязка

Применим

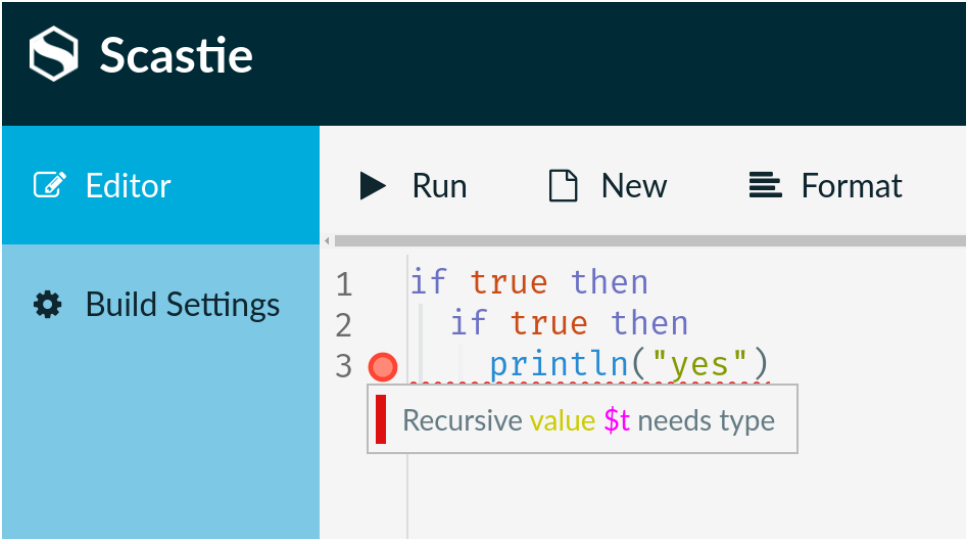
- ...и так далее
- Было найдено 9 подобных ошибок
  - В JIT-, AOT-оптимизаторе, тайпчекере, кодогенераторе
- Ещё 8 во время написания спецификации
- Наша спецификация
  - Подмножество
    - Завершающиеся программы
    - Циклы, ветвления, присваивания, исключения
    - Классы без методов, числа, массивы
  - ~330 строк кода спецификация языка + столько же обвязка
  - Частично деривированные, частично рукописные генераторы

Ещё?

Ещё?



Ещё?


Scastie
Editor
Build Settings
Run
New
Format
1 if true then
2 if true then
3 println("yes")
Recursive value $t needs type

- Property-based testing

- Property-based testing
  - ✔ *хороший* метод

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы
  - ✔ мощны и выразительны

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы
  - ✔ мощны и выразительны
  - ✘ высокий уровень входа

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы
  - ✔ мощны и выразительны
  - ✘ высокий уровень входа
  - ✘ нет в мейнстриме (пока?)

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы
  - ✔ мощны и выразительны
  - ✘ высокий уровень входа
  - ✘ нет в мейнстриме (пока?)
  - ✔ полезны не только в качестве спецификации

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы
  - ✔ мощны и выразительны
  - ✘ высокий уровень входа
  - ✘ нет в мейнстриме (пока?)
  - ✔ полезны не только в качестве спецификации
- Они вместе, как это ни удивительно, работают

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы
  - ✔ мощны и выразительны
  - ✘ высокий уровень входа
  - ✘ нет в мейнстриме (пока?)
  - ✔ полезны не только в качестве спецификации
- Они вместе, как это ни удивительно, работают
  - ✔ позволяют тестировать нетестируемое

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы
  - ✔ мощны и выразительны
  - ✘ высокий уровень входа
  - ✘ нет в мейнстриме (пока?)
  - ✔ полезны не только в качестве спецификации
- Они вместе, как это ни удивительно, работают
  - ✔ позволяют тестировать нетестируемое
  - ✘ инструментальная поддержка на зачаточном уровне

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы
  - ✔ мощны и выразительны
  - ✘ высокий уровень входа
  - ✘ нет в мейнстриме (пока?)
  - ✔ полезны не только в качестве спецификации
- Они вместе, как это ни удивительно, работают
  - ✔ позволяют тестировать нетестируемое
  - ✘ инструментальная поддержка на зачаточном уровне
  - ✘ методы спецификации ещё не отработаны

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы
  - ✔ мощны и выразительны
  - ✘ высокий уровень входа
  - ✘ нет в мейнстриме (пока?)
  - ✔ полезны не только в качестве спецификации
- Они вместе, как это ни удивительно, работают
  - ✔ позволяют тестировать нетестируемое
  - ✘ инструментальная поддержка на зачаточном уровне
  - ✘ методы спецификации ещё не отработаны
  - ✘ есть проблемы со скоростью работы

- Property-based testing
  - ✔ *хороший* метод
  - ✘ требует освоения
  - ✘ требует серьёзного взгляда на требования и формализацию
  - ✔ экономически оправдан для сложных и ответственных систем
- Зависимые типы
  - ✔ мощны и выразительны
  - ✘ высокий уровень входа
  - ✘ нет в мейнстриме (пока?)
  - ✔ полезны не только в качестве спецификации
- Они вместе, как это ни удивительно, работают
  - ✔ позволяют тестировать нетестируемое
  - ✘ инструментальная поддержка на зачаточном уровне
  - ✘ методы спецификации ещё не отработаны
  - ✘ есть проблемы со скоростью работы
  - ✔ мы только в начале пути

Если стало интересно

Эта презентация

DepTyCheck, примеры

Код со слайдов

Спасибо!

Эта презентация

Код со слайдов

DepTyCheck, примеры

Вопросы?