

# Байки от зависимых типов

## Типов, зависимых от зависимых типов

Денис Буздалов

30 ноября 2023

Привет  
●

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○○

Напоследок  
○

# Дисклеймеры

Привет



Mundane use

oooo

Отступление: PBT

ooooooo

DepTyCheck: знакомство

oooooooo

DepTyCheck: жизнь

oooooooo

Напоследок

o

# Дисклеймеры

- Доклад всё ещё познавательно-развлекательный

# Дисклеймеры

- Доклад всё ещё познавательно-развлекательный
- Но никакой жалости к слушателям ;-)

# Дисклеймеры

- Доклад всё ещё познавательно-развлекательный
- Но никакой жалости к слушателям ;-)
  - Код почти только на Idris
  - Более сложные зависимые типы

# Дисклеймеры

- Доклад всё ещё познавательно-развлекательный
- Но никакой жалости к слушателям ;-)
  - Код почти только на Idris
  - Более сложные зависимые типы
- Заметание под ковёр деталей!

# Дисклеймеры

- Доклад всё ещё познавательно-развлекательный
- Но никакой жалости к слушателям ;-)
  - Код почти только на Idris
  - Более сложные зависимые типы
- Заметание под ковёр деталей!
- Никакой полноты представления



@edwinb@types.pl 🔒

@edwinbrady



A mundane use of dependent types... I just tried to add an environment variable to Idris 2, and it wouldn't compile because I forgot to add it to the --help output.

\*offers compiler a jelly baby\*

17:23 · 28 февр. 21



Привет  
○

Mundane use  
○●○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○○

Напоследок  
○

# Как так получилось?

# Как так получилось?

```
idrisGetEnv : HasIO io => (name : String) ->
  (known : IsJust $ find (name ==) $ (.name) <$> Envs) =>
  io (Maybe String)
```

## Как так получилось?

```
Envs : List EnvDesc
Envs = [ -- <name>           <help>
  MkEnvDesc "EDITOR"         "Editor used in REPL :e command",
  MkEnvDesc "IDRIS2_PREFIX"  "Idris2 installation prefix",
  ...
  MkEnvDesc "IDRIS2_PACKAGE_PATH" "Where to look for packages",
  ...
  MkEnvDesc "NO_COLOR"        "Instruct Idris not to print color"]
```

```
idrisGetEnv : HasIO io => (name : String) ->
  (0 known : IsJust $ find (name ==) $ (.name) <$> Envs) =>
  io (Maybe String)
```

## Как так получилось?

```
Envs : List EnvDesc
Envs = [ -- <name>           <help>
  MkEnvDesc "EDITOR"         "Editor used in REPL :e command",
  MkEnvDesc "IDRIS2_PREFIX"  "Idris2 installation prefix",
  ...
  MkEnvDesc "IDRIS2_PACKAGE_PATH" "Where to look for packages",
  ...
  MkEnvDesc "NO_COLOR"        "Instruct Idris not to print color"]

data IsJust : Maybe a → Type where
  ItIsJust : IsJust (Just x)

idrisGetEnv : HasIO io ⇒ (name : String) →
  (0 known : IsJust $ find (name ==) $ (.name) <$> Envs) ⇒
  io (Maybe String)
```

# Как GADT, только вкуснее

```
-- data IsJust : Maybe a → Type where
--   ItIsJust : IsJust (Just x)
```

# Как GADT, только вкуснее

```
--           data IsJust : Maybe a → Type where
--           ItIsJust : IsJust (Just x)
```

```
enum IsList[A]:
  case ItIsList[A]() extends IsList[List[A]]
```

## Как GADT, только вкуснее

```
--           data IsJust : Maybe a → Type where
--           ItIsJust : IsJust (Just x)
```

```
enum IsList[A]:
  case ItIsList[A]() extends IsList[List[A]]
```

```
data IsList : Type → Type where
  ItIsList : IsList $ List a
```

## Как GADT, только вкуснее

```
--                                data IsJust : Maybe a → Type where
--                                ItIsJust : IsJust (Just x)
```

```
enum IsList[A]:
  case ItIsList[A]() extends IsList[List[A]]
def ff[A](using li: IsList[A])(xs : A): Int =
  li match { case ItIsList() ⇒ xs.length }
```

```
data IsList : Type → Type where
  ItIsList : IsList $ List a
```



## Как GADT, только вкуснее

```
--                                data IsJust : Maybe a → Type where
--                                ItIsJust : IsJust (Just x)
```

```
enum IsList[A]:
  case ItIsList[A]() extends IsList[List[A]]

def ff[A](using li: IsList[A])(xs : A): Int =
  li match { case ItIsList() ⇒ xs.length }
```

```
data IsList : Type → Type where
  ItIsList : IsList $ List a

ff : IsList a ⇒ a → Nat
ff @{ItIsList} xs = length xs
```

Привет  
○

Mundane use  
○○○●

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○○

Напоследок  
○

# Безопасное использование

# Безопасное использование

```
-- idrisGetEnv : HasIO io => (name : String) ->
--   (θ _ : IsJust $ find (name ==) $ (.name) <$> Envs) =>
--   io (Maybe String)
```

# Безопасное использование

```
--      idrisGetEnv : HasIO io => (name : String) ->
--      (θ _ : IsJust $ find (name ==) $ (.name) <$> Envs) =>
--      io (Maybe String)
```

```
updateEnv : MonadState Conf io => HasIO io => io ()
updateEnv = do
  ... something
  bpath <- idrisGetEnv "EDITOR"
  whenJust bpath setEditor
  ... something more

... etc
```

## Безопасное использование

```
--      idrisGetEnv : HasIO io => (name : String) ->
--      (θ _ : IsJust $ find (name ==) $ (.name) <$> Envs) =>
--      io (Maybe String)
```

```
updateEnv : MonadState Conf io => HasIO io => io ()
updateEnv = do
  ... something
  bpath <- idrisGetEnv "EDITOR"
  whenJust bpath setEditor
  ... something more
  pdirs <- idrisGetEnv "IDRIS2_PACKAGE_PATH"
  whenJust pdirs $ traverse_ addPackageDir . splitPaths
  ... etc
```

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
●○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○

Напоследок  
○

# 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
- (Иногда) автоматическая деривация генераторов

# Property-based testing

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

# Property-based testing

- Тестирование функции/системы на *произвольном* входе
- Оценка, а не предугадывание результата
- Рандомизированная генерация входных значений
- (Иногда) автоматическая деривация генераторов
- (Иногда) автоматический shrinking
- QuickCheck, Hedgehog, Validity, ScalaCheck, ...

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○●○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○

Напоследок  
○

# Property-based testing

# Property-based testing

```
sqrtOk : Property
sqrtOk = property $ do
  sqrt 4.0 == 2.0
```

# Property-based testing

```
sqrtOk : Property
sqrtOk = property $ do
  x <- forAll ?gen
  sqrt 4.0 == 2.0
```

# Property-based testing

```
nonNegative : Gen Double
```

```
sqrtOk : Property
```

```
sqrtOk = property $ do
```

```
  x <- forAll ?gen
```

```
  sqrt 4.0 == 2.0
```



# Property-based testing

```
nonNegative : Gen Double  
  
sqrtOk : Property  
sqrtOk = property $ do  
  x <- forAll nonNegative  
  sqrt 4.0 == 2.0
```

# Property-based testing

```
nonNegative : Gen Double  
  
sqrtOk : Property  
sqrtOk = property $ do  
  x <- forAll nonNegative  
  sqrt x == ?what
```

# Property-based testing

```
nonNegative : Gen Double  
sqrtOk : Property  
sqrtOk = property $ do  
  x <- forAll nonNegative  
  sqrt x * sqrt x =~ x
```

# Property-based testing

```
nonNegative : Gen Double
```

```
sqrtOk : Property
```

```
sqrtOk = property $ do  
  x <- forAll nonNegative  
  sqrt x * sqrt x =~ x
```

```
someList : Gen $ List Nat
```

```
reverseAndConcat : Property
```

```
reverseAndConcat = property $ do  
  (xs, ys) <- [| (forAll someList, forAll someList) |]  
  reverse xs ++ reverse ys == reverse (ys ++ xs)
```

## А что, если `reverse = id`?

```
nonNegative : Gen Double
```

```
sqrtOk : Property
```

```
sqrtOk = property $ do  
  x <- forAll nonNegative  
  sqrt x * sqrt x =~ x
```

```
someList : Gen $ List Nat
```

```
reverseAndConcat : Property
```

```
reverseAndConcat = property $ do  
  (xs, ys) <- [| (forAll someList, forAll someList) |]  
  reverse xs ++ reverse ys == reverse (ys ++ xs)
```

А что, если `reverse = id`?

× `reverseAndConcat` failed after 13 tests.

```
forAll 0 =  
  [0]
```

```
forAll 1 =  
  [1]
```

```
----- Failed (- lhs) (+ rhs) -----  
- "[0, 1]"  
+ "[1, 0]"
```

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○●○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○

Напоследок  
○

# Не только функции

Привет  
○

Mundane use  
○○○○

Отступление: РВГ  
○○○●○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○

Напоследок  
○

## Пример: системы с состоянием, базы данных



# Пример: системы с состоянием, базы данных

```
data DBQuery : Type → Type where
  CreateTable : String → DBQuery Unit
  Insert      : String → List String → DBQuery Unit
  Lookup      : String → DBQuery $ List String
```

## Пример: системы с состоянием, базы данных

```
data DBQuery : Type → Type where
  CreateTable : String → DBQuery Unit
  Insert      : String → List String → DBQuery Unit
  Lookup      : String → DBQuery $ List String
query : Gen $ Exists DBQuery
```

# Пример: системы с состоянием, базы данных

```
data DBQuery : Type → Type where
  CreateTable : String → DBQuery Unit
  Insert      : String → List String → DBQuery Unit
  Lookup      : String → DBQuery $ List String

query : Gen $ Exists DBQuery

interface StMachine st where
  doStep : DBQuery res → st → (st, res)
```

## Пример: системы с состоянием, базы данных

```
data DBQuery : Type → Type where
  CreateTable : String → DBQuery Unit
  Insert      : String → List String → DBQuery Unit
  Lookup      : String → DBQuery $ List String

query : Gen $ Exists DBQuery
```

```
interface StMachine st where
  doStep : DBQuery res → st → (st, res)
```

```
StMachine ModelState where
  doStep q s = ?expected_behaviour
```

```
StMachine RealState where
  doStep q s = ?call_to_real_db
```

## Пример: системы с состоянием, базы данных

`conformSeq` : (tr1 : StMachine st1) → (init1 : st1) →  
              (tr2 : StMachine st2) → (init2 : st2) → Property

## Пример: системы с состоянием, базы данных

```
conformSeq : (tr1 : StMachine st1) → (init1 : st1) →  
              (tr2 : StMachine st2) → (init2 : st2) → Property  
conformSeq tr1 init1 tr2 init2 = property $ do  
  qs <- forAll $ list querySizeRange query
```

## Пример: системы с состоянием, базы данных

```
conformSeq : (tr1 : StMachine st1) → (init1 : st1) →  
              (tr2 : StMachine st2) → (init2 : st2) → Property  
conformSeq tr1 init1 tr2 init2 = property $ do  
  qs <- forAll $ list querySizeRange query  
  conformQs qs init1 init2  
where  
  conformQs : List (Exists DBQuery) →  
              st1 → st2 → PropertyT ()
```

## Пример: системы с состоянием, базы данных

```
conformSeq : (tr1 : StMachine st1) → (init1 : st1) →  
              (tr2 : StMachine st2) → (init2 : st2) → Property  
conformSeq tr1 init1 tr2 init2 = property $ do  
  qs <- forAll $ list querySizeRange query  
  conformQs qs init1 init2  
where  
  conformQs : List (Exists DBQuery) →  
              st1 → st2 → PropertyT ()  
  conformQs [] _ _ = success
```



## Пример: системы с состоянием, базы данных

```
conformSeq : (tr1 : StMachine st1) → (init1 : st1) →  
              (tr2 : StMachine st2) → (init2 : st2) → Property  
conformSeq tr1 init1 tr2 init2 = property $ do  
  qs <- forAll $ list querySizeRange query  
  conformQs qs init1 init2  
where  
  conformQs : List (Exists DBQuery) →  
              st1 → st2 → PropertyT ()  
conformQs [] _ _ = success  
conformQs (Evidence _ q::qs) s1 s2 = do  
  let (s1', res1) = doStep @{tr1} q s1  
  let (s2', res2) = doStep @{tr2} q s2
```

## Пример: системы с состоянием, базы данных

```
conformSeq : (tr1 : StMachine st1) → (init1 : st1) →  
              (tr2 : StMachine st2) → (init2 : st2) → Property  
conformSeq tr1 init1 tr2 init2 = property $ do  
  qs <- forAll $ list querySizeRange query  
  conformQs qs init1 init2  
where  
  conformQs : List (Exists DBQuery) →  
              st1 → st2 → PropertyT ()  
conformQs [] _ _ = success  
conformQs (Evidence _ q::qs) s1 s2 = do  
  let (s1', res1) = doStep @{tr1} q s1  
  let (s2', res2) = doStep @{tr2} q s2  
  res1 == res2
```

## Пример: системы с состоянием, базы данных

```
conformSeq : (tr1 : StMachine st1) → (init1 : st1) →  
              (tr2 : StMachine st2) → (init2 : st2) → Property  
conformSeq tr1 init1 tr2 init2 = property $ do  
  qs <- forAll $ list querySizeRange query  
  conformQs qs init1 init2  
where  
  conformQs : List (Exists DBQuery) →  
              st1 → st2 → PropertyT ()  
conformQs [] _ _ = success  
conformQs (Evidence _ q::qs) s1 s2 = do  
  let (s1', res1) = doStep @{tr1} q s1  
  let (s2', res2) = doStep @{tr2} q s2  
  res1 == res2  
  conformQs qs s1' s2'
```

## Пример: системы с состоянием, базы данных

```
conformPar : (tr1 : StMachine st1) → (init1 : st1) →  
              (tr2 : StMachine st2) → (init2 : st2) → Property  
conformPar tr1 init1 tr2 init2 = property $ do  
  prefixQs <- forAll $ list querySizeRange query  
  parQs <- forAll $ list1 thrs $ list querySizeRange query
```

## Пример: системы с состоянием, базы данных

```
conformPar : (tr1 : StMachine st1) → (init1 : st1) →  
              (tr2 : StMachine st2) → (init2 : st2) → Property  
conformPar tr1 init1 tr2 init2 = property $ do  
  prefixQs <- forAll $ list querySizeRange query  
  parQs <- forAll $ list1 thrs $ list querySizeRange query  
  conformQs prefixQs parQs init1 init2  
where  
  conformQs : List (Exists DBQuery) →  
              List1 (List (Exists DBQuery)) →  
              _ → _ → _  
...
```



Clojure/West

March 24-26 2014  
The Palace Hotel San Francisco

## Bug #4

**Prefix:**

```
open_file(dets_table, [{type, bag}]) --> dets_table  
close(dets_table) --> ok  
open_file(dets_table, [{type, bag}]) --> dets_table
```

**Parallel:**

1. lookup(dets\_table, 0) --> []
2. insert(dets\_table, {0, 0}) --> ok
3. insert(dets\_table, {0, 0}) --> ok

**Result: ok****premature eof**

John Hughes - Testing the Hard Stuff and Staying Sane

<sup>1</sup><https://www.youtube.com/watch?v=zi0rHwfiX1Q>

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
●○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○○

Напоследок  
○

# Сложные входные данные

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
●○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○○

Напоследок  
○

# Сложные входные данные

- Для компиляторов



Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
●○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○○

Напоследок  
○

# Сложные входные данные

- Для компиляторов, например
  - Тайпчекеры

# Сложные входные данные

- Для компиляторов, например
  - Тайпчекеры
  - Оптимизаторы

# Сложные входные данные

- Для компиляторов, например
  - Тайпчекеры
  - Оптимизаторы
  - Кодогенераторы

## Сложные входные данные

- Для компиляторов, например
  - Тайпчекеры
  - Оптимизаторы
  - Кодогенераторы
- Для рантаймов типа JVM

## Сложные входные данные

- Для компиляторов, например
  - Тайпчекеры
  - Оптимизаторы
  - Кодогенераторы
- Для рантаймов типа JVM
  - AOT, JIT

## Сложные входные данные

- Для компиляторов, например
  - Тайпчекеры
  - Оптимизаторы
  - Кодогенераторы
- Для рантаймов типа JVM
  - AOT, JIT
- Драйвера, сервисы, ...
- ...

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
●○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○○

Напоследок  
○

## Зависимые типы?

## Зависимые типы! Помните?

```
data SortedBinTree : Type → Type
data All : (a → Bool) → SortedBinTree a → Type

data SortedBinTree : Type → Type where
  Empty : SortedBinTree a
  Node   : Ord a ⇒ (x : a) → (left, right : SortedBinTree a) →
    All (< x) left ⇒ All (x <) right ⇒ SortedBinTree a

data All : (a → Bool) → SortedBinTree a → Type where
  Empty' : All prop Empty
  Node'   : (o : Ord a) ⇒ {o prop : a → Bool} →
    {o p1 : All (< x) l} → {o pr : All (x <) r} →
    So (prop x) → All prop l → All prop r →
    All prop $ Node x l r @{o} @{p1} @{pr}
```



Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○●○○○○○

DepTyCheck: жизнь  
○○○○○○○○○

Напоследок  
○

# Зависимые типы + property-based testing

- Зависимые типы для формирования сложных входных данных

## Зависимые типы + property-based testing

- Зависимые типы для формирования сложных входных данных
- Описать тип так, чтобы любое значение подходило

## Зависимые типы + property-based testing

- Зависимые типы для формирования сложных входных данных
- Описать тип так, чтобы любое значение подходило
- Возможность автоматической деривации генератора

## Зависимые типы + property-based testing

- Зависимые типы для формирования сложных входных данных
- Описать тип так, чтобы любое значение подходило
- Возможность автоматической деривации генератора
- ...
- Profit!

## Пример: очень маленький язык

```
data Stmts : (funs  : List (Name, FunSig)) →  
              (prev  : List (Name, Type)) →  
              (postV : List (Name, Type)) → Type where
```

## Пример: очень маленький язык

```
data Stmts : (funcs  : List (Name, FunSig)) →  
              (prev   : List (Name, Type)) →  
              (postV  : List (Name, Type)) → Type where
```

```
(>>) : Stmts funcs prev midV → Stmts funcs midV postV →  
      Stmts funcs prev postV
```

## Пример: очень маленький язык

```
data Stmts : (funcs  : List (Name, FunSig)) →  
              (preV   : List (Name, Type)) →  
              (postV  : List (Name, Type)) → Type where  
  
(.) : (ty : Type) → (n : Name) →  
      Stmts funcs vars ((n, ty)::vars)
```

```
(>>) : Stmts funcs preV midV → Stmts funcs midV postV →  
      Stmts funcs preV postV
```

## Пример: очень маленький язык

```
data Stmts : (funcs  : List (Name, FunSig)) →  
              (prev   : List (Name, Type)) →  
              (postV  : List (Name, Type)) → Type where
```

```
(.) : (ty : Type) → (n : Name) →  
      Stmts funcs vars ((n, ty)::vars)
```

```
(#=) : (n : Name) →  
       (v : Expr funcs vars ?expr_ty) →  
       Stmts funcs vars vars
```

```
(>>) : Stmts funcs prev midV → Stmts funcs midV postV →  
      Stmts funcs prev postV
```



## Пример: очень маленький язык

```
data Stmts : (funcs  : List (Name, FunSig)) →  
              (prev   : List (Name, Type)) →  
              (postV  : List (Name, Type)) → Type where  
  
(.) : (ty : Type) → (n : Name) →  
      Stmts funcs vars ((n, ty)::vars)  
  
(#)=) : (n : Name) → (0 lk : IsJust $ lookup n vars) ⇒  
        (v : Expr funcs vars lk.found) →  
        Stmts funcs vars vars  
  
(>>) : Stmts funcs prev midV → Stmts funcs midV postV →  
      Stmts funcs prev postV
```

## Пример: очень маленький язык

```
data Stmts : (funcs  : List (Name, FunSig)) →  
              (prev   : List (Name, Type)) →  
              (postV  : List (Name, Type)) → Type where  
  
(.)  : (ty : Type) → (n : Name) →  
        Stmts funcs vars ((n, ty)::vars)  
  
(#=) : (n : Name) → (0 lk : IsJust $ lookup n vars) ⇒  
        (v : Expr funcs vars lk.found) →  
        Stmts funcs vars vars  
  
If    : (cond : Expr funcs vars Bool) →  
        Stmts funcs vars vThen → Stmts funcs vars vElse →  
        Stmts funcs vars vars  
  
(>>) : Stmts funcs prev midV → Stmts funcs midV postV →  
        Stmts funcs prev postV
```

## Пример: очень маленький язык

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

## Пример: очень маленький язык

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

```
C : (x : ty) → Expr funs vars ty
```

## Пример: очень маленький язык

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

```
C : (x : ty) → Expr funs vars ty
```

```
V : (n : Name) → (0 lk : IsJust $ lookup n vars) ⇒  
  Expr funs vars lk.found
```

## Пример: очень маленький язык

```
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 : IsJust $ lookup n vars) ⇒
   Expr funs vars lk.found
```

## Пример: очень маленький язык

```
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 : IsJust $ lookup n vars) ⇒
  Expr funs vars lk.found
```

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

# Пример: очень маленький язык

```
program : Stmt [] [] ?  
program = do  
  Int. "x"  
  "x" #= C 5
```



# Пример: очень маленький язык

```
program : Stmt [] [] ?  
program = do  
  Int. "x"  
  "x" #= C 5  
  Int. "y"; Bool. "res"
```

# Пример: очень маленький язык

```
program : StmtS [] [] ?  
program = do  
  Int. "x"  
  "x" #= C 5  
  Int. "y"; Bool. "res"
```

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

## Пример: очень маленький язык

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

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

## Пример: очень маленький язык

```
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]
```

## Пример: очень маленький язык

```
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"])
    ?then_branch
    ?else_branch
```

## Пример: очень маленький язык

```
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)
    ?else_branch
```

## Пример: очень маленький язык

```
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]])
```

## Пример: очень маленький язык

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



## Пример: очень маленький язык

```
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"]
```

## Пример: очень маленький язык

```
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"]
```

## Пример: очень маленький язык

```
failing "Can't find an implementation for IsJust"
bad : Stmts StdF [] ?
bad = do
  Int. "x"; "x" #= C 5
  "z" #= V "x"
```

## Пример: очень маленький язык

```
failing "Can't find an implementation for IsJust"
bad : Stmts StdF [] ?
bad = do
  Int. "x"; "x" #= C 5
  "z" #= V "x"
```

```
failing "Can't find an implementation for IsJust"
bad : Stmts StdF [] ?
bad = do
  Int. "x"
  "x" #= V "z"
```

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: **жизнь**  
●○○○○○○○

Напоследок  
○

# Пример из жизни

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
●○○○○○○○

Напоследок  
○

## 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
- Специфировали подмножество

## Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Специфировали подмножество
  - Завершающиеся программы

## Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Специфировали подмножество
  - Завершающиеся программы
  - Циклы, ветвления, присваивания, исключения

## Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Специфировали подмножество
  - Завершающиеся программы
  - Циклы, ветвления, присваивания, исключения
  - Классы без методов, числа, массивы

## Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Специфировали подмножество
  - Завершающиеся программы
  - Циклы, ветвления, присваивания, исключения
  - Классы без методов, числа, массивы
- Наша спецификация

## Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Специфировали подмножество
  - Завершающиеся программы
  - Циклы, ветвления, присваивания, исключения
  - Классы без методов, числа, массивы
- Наша спецификация
  - Описание семантически корректных программ из подмножества

## Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Специфировали подмножество
  - Завершающиеся программы
  - Циклы, ветвления, присваивания, исключения
  - Классы без методов, числа, массивы
- Наша спецификация
  - Описание семантически корректных программ из подмножества
  - ~330 строк кода, аналогичного `Stmts` и `Expr` + обвязка

## Пример из жизни

- Диалект Typescript
- Имеет интерпретатор с JIT и компилятор
- Свежая промышленная разработка
- Специфировали подмножество
  - Завершающиеся программы
  - Циклы, ветвления, присваивания, исключения
  - Классы без методов, числа, массивы
- Наша спецификация
  - Описание семантически корректных программ из подмножества
  - ~330 строк кода, аналогичного `Stmts` и `Expr` + обвязка
  - Частично деривированные, частично рукописные генераторы



Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○●○○○○○○

Напоследок  
○

# Пример из жизни

Testing

---

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○●○○○○○○

Напоследок  
○

## Пример из жизни

Testing.

---

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○●○○○○○○

Напоследок  
○

## Пример из жизни

Testing..

---

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○●○○○○○○

Напоследок  
○

## Пример из жизни

Testing...

---

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○●○○○○○○

Напоследок  
○

# Пример из жизни

Testing

---

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○●○○○○○○

Напоследок  
○

## Пример из жизни

Testing.

---

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○●○○○○○○

Напоследок  
○

## Пример из жизни

Testing..

---

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○●○○○○○○

Напоследок  
○

## Пример из жизни

Testing...

---

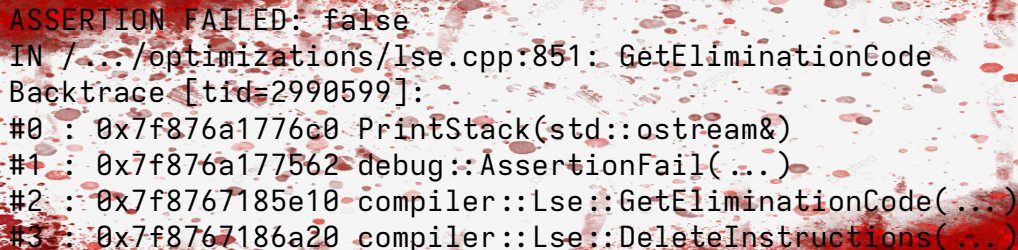


## Пример из жизни

```
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(...)
...
```

---

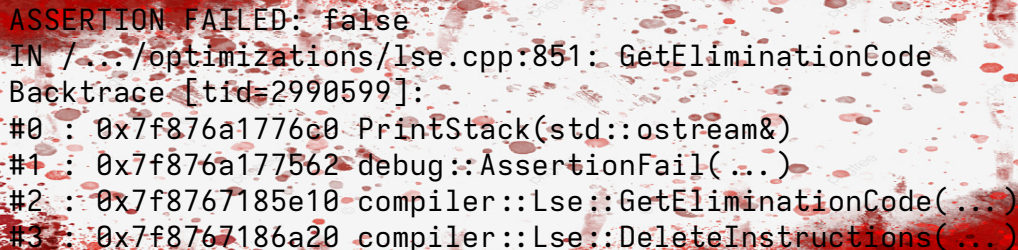
## Пример из жизни



```
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(...)
```

---

## Пример из жизни



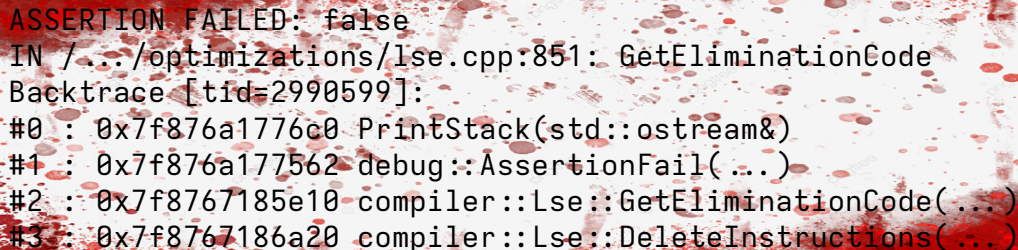
```
ASSERTION FAILED: false
IN /.../optimizations/lse.cpp:851: GetEliminationCode
Backtrace [tid=2990599]:
#0 : 0x7f876a1776c0 PrintStack(std::ostream&)
#1 : 0x7f876a177562 debug::AssertionFail(...)
#2 : 0x7f8767185e10 compiler::Lse::GetEliminationCode(...)
#3 : 0x7f8767186a20 compiler::Lse::DeleteInstructions(...)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

Shrinking<sup>1</sup>

---

<sup>1</sup> пока вручную или внешним инструментом

## Пример из жизни



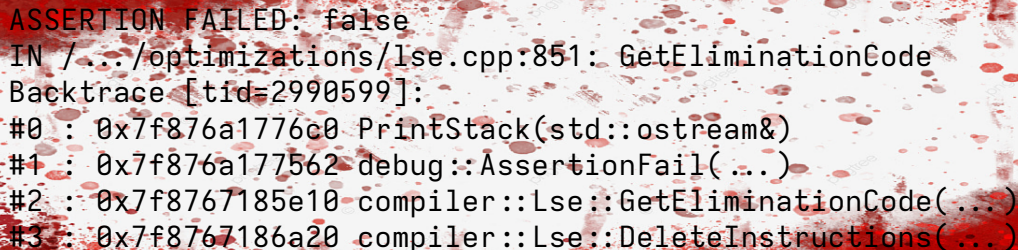
```
ASSERTION FAILED: false
IN /.../optimizations/lse.cpp:851: GetEliminationCode
Backtrace [tid=2990599]:
#0 : 0x7f876a1776c0 PrintStack(std::ostream&)
#1 : 0x7f876a177562 debug::AssertionFail(...)
#2 : 0x7f8767185e10 compiler::Lse::GetEliminationCode(...)
#3 : 0x7f8767186a20 compiler::Lse::DeleteInstructions(...)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

Shrinking<sup>1</sup>.

---

<sup>1</sup> пока вручную или внешним инструментом

## Пример из жизни



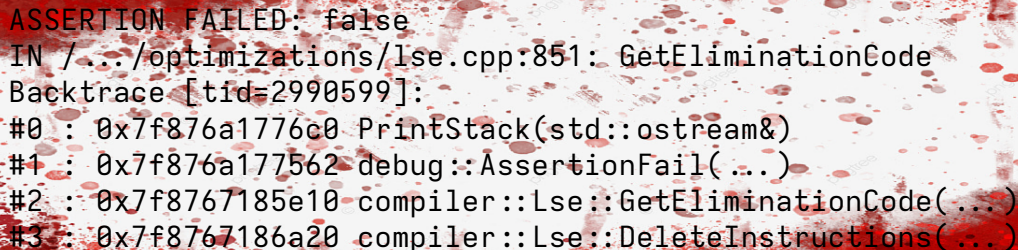
```
ASSERTION FAILED: false
IN /.../optimizations/lse.cpp:851: GetEliminationCode
Backtrace [tid=2990599]:
#0 : 0x7f876a1776c0 PrintStack(std::ostream&)
#1 : 0x7f876a177562 debug::AssertionFail(...)
#2 : 0x7f8767185e10 compiler::Lse::GetEliminationCode(...)
#3 : 0x7f8767186a20 compiler::Lse::DeleteInstructions(...)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

Shrinking<sup>1</sup>..

---

<sup>1</sup> пока вручную или внешним инструментом

## Пример из жизни



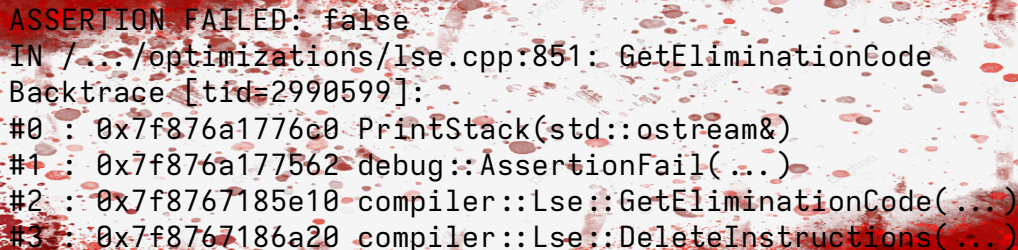
```
ASSERTION FAILED: false
IN /.../optimizations/lse.cpp:851: GetEliminationCode
Backtrace [tid=2990599]:
#0 : 0x7f876a1776c0 PrintStack(std::ostream&)
#1 : 0x7f876a177562 debug::AssertionFail(...)
#2 : 0x7f8767185e10 compiler::Lse::GetEliminationCode(...)
#3 : 0x7f8767186a20 compiler::Lse::DeleteInstructions(...)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

Shrinking<sup>1</sup>...

---

<sup>1</sup> пока вручную или внешним инструментом

## Пример из жизни



```
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(...)
```

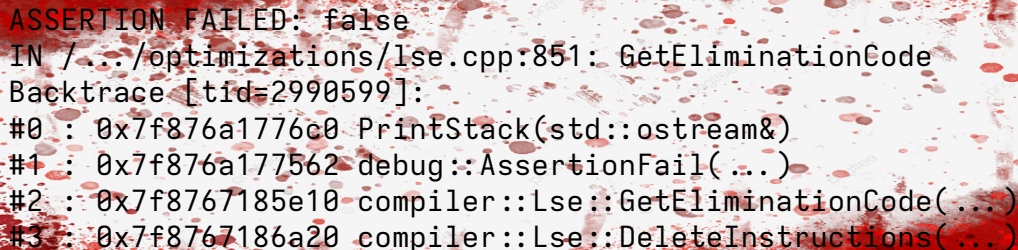
Shrinking<sup>1</sup>

---

<sup>1</sup> пока вручную или внешним инструментом



## Пример из жизни



```
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(...)
```

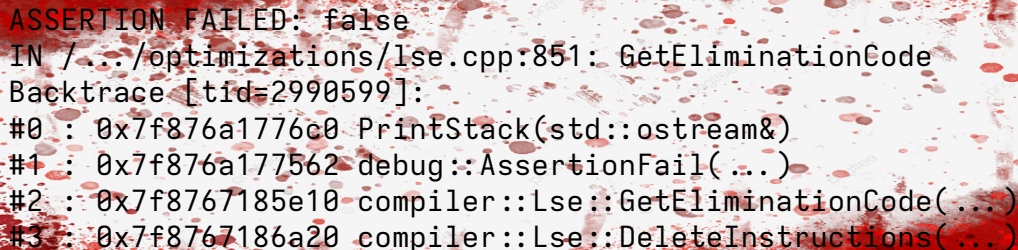
Shrinking<sup>1</sup>.

---

<sup>1</sup> пока вручную или внешним инструментом



## Пример из жизни



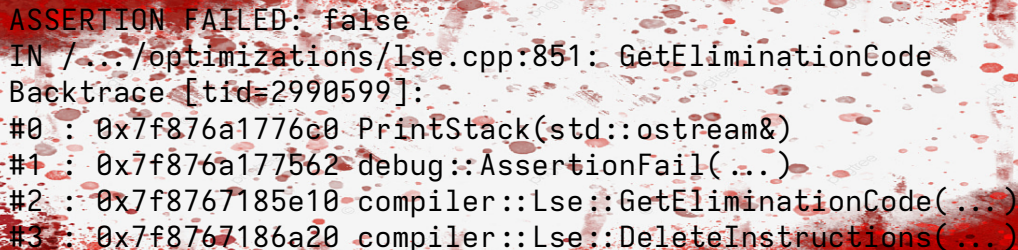
```
ASSERTION FAILED: false
IN /.../optimizations/lse.cpp:851: GetEliminationCode
Backtrace [tid=2990599]:
#0 : 0x7f876a1776c0 PrintStack(std::ostream&)
#1 : 0x7f876a177562 debug::AssertionFail(...)
#2 : 0x7f8767185e10 compiler::Lse::GetEliminationCode(...)
#3 : 0x7f8767186a20 compiler::Lse::DeleteInstructions(...)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

Shrinking<sup>1</sup>..

---

<sup>1</sup> пока вручную или внешним инструментом

## Пример из жизни



```
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<sup>1</sup>...

---

<sup>1</sup> пока вручную или внешним инструментом

## Пример из жизни

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

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○●○○○

Напоследок  
○

# Пример из жизни

Testing

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○●○○○

Напоследок  
○

## Пример из жизни

Testing.

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○●○○○

Напоследок  
○

## Пример из жизни

Testing..

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○●○○○

Напоследок  
○

## Пример из жизни

Testing...

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○●○○○

Напоследок  
○

# Пример из жизни

Testing



Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○●○○○

Напоследок  
○

## Пример из жизни

Testing.

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○●○○○

Напоследок  
○

## Пример из жизни

Testing..

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○●○○○

Напоследок  
○

## Пример из жизни

Testing...

## Пример из жизни

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()

...

## Пример из жизни

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

## Пример из жизни

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.

## Пример из жизни

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..

## Пример из жизни

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...



## Пример из жизни

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

## Пример из жизни

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.

## Пример из жизни

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..

## Пример из жизни

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...

## Пример из жизни

```
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  
}
```

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○●○○

Напоследок  
○

# Пример из жизни

Testing

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○●○○

Напоследок  
○

## Пример из жизни

Testing.

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○●○○

Напоследок  
○

## Пример из жизни

Testing..



Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○●○○

Напоследок  
○

## Пример из жизни

Testing...

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○●○○

Напоследок  
○

# Пример из жизни

Testing

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○●○○

Напоследок  
○

## Пример из жизни

Testing.

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○●○○

Напоследок  
○

## Пример из жизни

Testing..

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○●○○

Напоследок  
○

## Пример из жизни

Testing...

## Пример из жизни

```
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(...)
...
```

## Пример из жизни

```
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

## Пример из жизни

```
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.



## Пример из жизни

```
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..

## Пример из жизни

```
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...

## Пример из жизни

```
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

## Пример из жизни

```
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.

## Пример из жизни

```
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..

## Пример из жизни

```
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...

## Пример из жизни

```
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))  
  }  
}
```

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○○

DepTyCheck: жизнь  
○○○○○○○○●

Напоследок  
○

# Выводы



Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○○○●

Напоследок  
○

## Выводы

- Ошибки у вполне тестированных систем лежат в пересечении множества фич

Привет  
○

Mundane use  
○○○○

Отступление: PBT  
○○○○○○○

DepTyCheck: знакомство  
○○○○○○○○

DepTyCheck: жизнь  
○○○○○○○●

Напоследок  
○

## Выводы

- Ошибки у вполне тестированных систем лежат в пересечении множества фич
- Одна спецификация находит много ошибок

## Выводы

- Ошибки у вполне тестированных систем лежат в пересечении множества фич
- Одна спецификация находит много ошибок
- Нацеленность не всегда полезна, ошибки могут быть не там, где мы их ожидаем увидеть

## Выводы

- Ошибки у вполне тестированных систем лежат в пересечении множества фич
- Одна спецификация находит много ошибок
- Нацеленность не всегда полезна, ошибки могут быть не там, где мы их ожидаем увидеть
- Property-based testing классный

## Выводы

- Ошибки у вполне тестированных систем лежат в пересечении множества фич
- Одна спецификация находит много ошибок
- Нацеленность не всегда полезна, ошибки могут быть не там, где мы их ожидаем увидеть
- Property-based testing классный
- Зависимые типы классные ;-)

## Если стало интересно



Код со слайдов



DepTyCheck



Idris 2 language



Эта презентация

# Спасибо!



Код со слайдов



DepTyCheck



Idris 2 language



Эта презентация

## Вопросы?