

# Мама и папа, вы видели, как я могу?

Код *точно* работает с внешними данными!

Денис Буздалов

JVM Day

Москва

30 августа 2025



Привет  
●

Ограничения в типах  
oooooooo

Деривация  
oooo

Внешний источник  
o

Заключение  
ooo

# О докладе

## О докладе

- Развлекательный доклад на сон грядущий ;-)

## О докладе

- Развлекательный доклад на сон грядущий ;-)
- Академическая среда и (не)практичность

## О докладе

- Развлекательный доклад на сон грядущий ;-)
- Академическая среда и (не)практичность
- Мало Scala, много непривычного кода

## О докладе

- Развлекательный доклад на сон грядущий ;-)
- Академическая среда и (не)практичность
- Мало Scala, много непривычного кода
- Будет сложно, вопросы на понимание приветствуются

## О докладе

- Развлекательный доклад на сон грядущий ;-)
- Академическая среда и (не)практичность
- Мало Scala, много непривычного кода
- Будет сложно, вопросы на понимание приветствуются
- *Очень* много деталей заматывается под ковёр, намеренное упрощение

Привет  
○

Ограничения в типах  
●○○○○○○○

Деривация  
○○○○

Внешний источник  
○

Заключение  
○○○

# Общение с внешним миром



# Общение с внешним миром

- Часто формально специфицировано
  - JsonSchema, Protobuf, ASN.1, другие IDL (interface description language)
  - сетевые протоколы, системные вызовы
  - описания политик (rego, casbin, ...)
  - ...

## Общение с внешним миром

- Часто формально специфицировано
  - JsonSchema, Protobuf, ASN.1, другие IDL (interface description language)
  - сетевые протоколы, системные вызовы
  - описания политик (rego, casbin, ...)
  - ...
- Single source of truth

# Общение с внешним миром

- Часто формально специфицировано
  - JsonSchema, Protobuf, ASN.1, другие IDL (interface description language)
  - сетевые протоколы, системные вызовы
  - описания политик (rego, casbin, ...)
  - ...
- Single source of truth
- Нахождение ошибок на этапе компиляции

# Общение с внешним миром

- Часто формально специфицировано
  - JsonSchema, Protobuf, ASN.1, другие IDL (interface description language)
  - сетевые протоколы, системные вызовы
  - описания политик (rego, casbin, ...)
  - ...
- Single source of truth
- Нахождение ошибок на этапе компиляции
  - Отражение спецификаций в языке

# Общение с внешним миром

- Часто формально специфицировано
  - JsonSchema, Protobuf, ASN.1, другие IDL (interface description language)
  - сетевые протоколы, системные вызовы
  - описания политик (rego, casbin, ...)
  - ...
- Single source of truth
- Нахождение ошибок на этапе компиляции
  - Отражение спецификаций в языке
- Востребованность
  - JING: <https://github.com/TomasMikula/jing>
  - ...

# Общение с внешним миром

- Часто формально специфицировано
  - **JsonSchema**, Protobuf, ASN.1, другие IDL (interface description language)
  - сетевые протоколы, системные вызовы
  - описания политик (rego, casbin, ...)
  - ...
- Single source of truth
- Нахождение ошибок на этапе компиляции
  - Отражение спецификаций в языке
- Востребованность
  - JING: <https://github.com/TomasMikula/jing>
  - ...

# Общение с внешним миром

```
{
  "type": "object",
  "properties": {
    "names": {
      "type": "array",
      "minItems": 2,
      "items": {
        "type": "string",
        "pattern": "[A-Z][A-Za-z\\-]*[A-Za-z]"
      }
    },
    "age": { "type": "integer", "minimum": 0 },
    "email": {
      "type": "string",
      "pattern": "[\\w·-]+@(?:[\\w-]+\\.)+\\w[\\w-]+"
    }
  }
}
```

## Какой использовать тип?

```
{
  "type": "object",
  "properties": {
    "names": {
      "type": "array",
      "minItems": 2,
      "items": {
        "type": "string",
        "pattern": "[A-Z][A-Za-z\\-]*[A-Za-z]"
      }
    },
    "age": { "type": "integer", "minimum": 0 },
    "email": {
      "type": "string",
      "pattern": "[\\w·-]+@(?:[\\w-]+\\.)+\\w[\\w-]+"
    }
  }
}
```



# Без ограничений

```
case class User(  
  names: List[String],  
  age: Int,  
  email: String)
```

// Scala

```
record User where  
  constructor MkUser  
  names : List String  
  age   : Int  
  email : String
```

— Idris

## Без ограничений

```
case class User(                                     // Scala
  names: List[String],
  age: Int,
  email: String)

val good =
  User(List("Denis", "Buzdalov"), 35, "test@example.com")
```

```
record User where                                   — Idris
  constructor MkUser
  names : List String
  age   : Int
  email : String

good = MkUser ["Denis", "Buzdalov"] 35 "test@example.com"
```

## Без ограничений

```
case class User(                                     // Scala
  names: List[String],
  age: Int,
  email: String)

val bad =
  User(List("denis"), -35, "bad@email")
```

```
record User where                                   — Idris
  constructor MkUser
  names : List String
  age   : Int
  email : String

bad = MkUser ["denis"] (-35) "bad@email"
```

## Smart-конструкторы

```
case class User private (                                     // Scala
  names: List[String], age: Int, email: String)
object User:
  def validate(names: List[String], age: Int,
    email: String): Option[User] = ???
```

```
export                                                         — Idris
record User where
  constructor MkUser
  names : List String; age : Int; email : String
mkUser : List String → Int → String → Maybe User
```

## Smart-конструкторы

```
case class User private (                               // Scala
    names: List[String], age: Int, email: String)

object User:
    def validate(names: List[String], age: Int,
        email: String): Option[User] = ???
```

## Smart-конструкторы

```
case class User private (                               // Scala
  names: List[String], age: Int, email: String)

object User:
  def validate(names: List[String], age: Int,
    email: String): Option[User] = ???

  def manageNames(names: List[String]) = ???
```

## Smart-конструкторы

```
case class User private (                                     // Scala
    names: List[String], age: Int, email: String)

object User:
    def validate(names: List[String], age: Int,
        email: String): Option[User] = ???

def manageNames(names: List[String]) = ???

// ...
val (firstName, lastName) = names match
    case first::(tail@(_::_)) => (first, tail.last)
    case _ => fail
val user = User.validate(names, age, email).getOrElse(fail)
val r = manageNames(names).getOrElse(fail)
// ...
```

# Parse, don't validate!

```
case class User private (                                     // Scala
  names: List[String], age: Int, email: String)

object User:
  def validate(names: List[String], age: Int,
    email: String): Option[User] = ???

def manageNames(names: List[String]) = ???

// ...
val (firstName, lastName) = names match
  case first::(tail@(_::_)) => (first, tail.last)
  case _ => fail
val user = User.validate(names, age, email).getOrElse(fail)
val r = manageNames(names).getOrElse(fail)
// ...
```



## Частичные структурные ограничения

```
case class User(                                     // Scala
  names: NonEmptyList2[String],
  age: Int,
  email: String)
```

## Частичные структурные ограничения

```
case class User(                                     // Scala
  names: NonEmptyList2[String],
  age: Int,
  email: String)
```

```
record User where                                   — Idris
  constructor MkUser
  names : Vect (2+n) String
  age   : Nat
  email : String
```

## Частичные структурные ограничения

```
case class User(                                     // Scala
  names: NonEmptyList2[String],
  age: Int,
  email: String)
```

```
record User where                                   — Idris
  constructor MkUser
  names : Vect (2+n) String
  age   : Nat
  email : String
```

```
failing "When unifying: Vect 0 String and: Vect (plus 1 ?n) String"
  bad = MkUser ["Denis"] 35 "bad@email"
```

Привет  
○

Ограничения в типах  
○○○○○●○○

Деривация  
○○○○

Внешний источник  
○

Заключение  
○○○

# Refined-типы

# Refined-типы

```
case class User(  
  names: List[String],  
  
  age: Int,  
  email: String)
```

```
// Scala
```

# Refined-типы

```
case class User(  
  names: List[String],  
  
  age: Int,  
  email: String)
```

```
// Scala
```

\* *Используется библиотека Iron, <https://github.com/Iltotore/iron>*

# Refined-типы

```
case class User(                                     // Scala
  names: List[String],

  age: Int :| Positive0,
  email: String)
```

\* *Используется библиотека Iron, <https://github.com/Iltotore/iron>*

# Refined-типы

```
case class User(                                     // Scala
  names: List[String] :|
    Length[GreaterEqual[2]],
  age: Int :| Positive0,
  email: String)
```

\* Используется библиотека *Iron*, <https://github.com/Iltotore/iron>



## Refined-типы

```
type nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]"
```

```
case class User(                                     // Scala
  names: List[String] :|
    Length[GreaterEqual[2]],
  age: Int :| Positive0,
  email: String)
```

\* Используется библиотека Iron, <https://github.com/Iltotore/iron>

## Refined-типы

```
type nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]"
```

```
case class User(                                     // Scala
  names: List[String] :|
    Length[GreaterEqual[2]] & ForAll[Match[nameRegex]],
  age: Int :| Positive0,
  email: String)
```

\* Используется библиотека Iron, <https://github.com/Iltotore/iron>

## Refined-типы

```
type nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]"
type emailRegex = "[\\w-\\.]+@(?:[\\w-]+\\.)+\\w[\\w-]+"

case class User(                                     // Scala
  names: List[String] :|
    Length[GreaterEqual[2]] & ForAll[Match[nameRegex]],
  age: Int :| Positive0,
  email: String)
```

\* Используется библиотека Iron, <https://github.com/Iltotore/iron>

## Refined-типы

```
type nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]"
type emailRegex = "[\\w-\\.]+@(?:[\\w-]+\\.)+\\w[\\w-]+"

case class User(                                     // Scala
  names: List[String] :|
    Length[GreaterEqual[2]] & ForAll[Match[nameRegex]],
  age: Int :| Positive0,
  email: String :| Match[emailRegex])
```

\* Используется библиотека Iron, <https://github.com/Iltotore/iron>

## Refined-типы

```
type nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]"
type emailRegex = "[\\w-\\.]+@([?:[\\w-]+\\.)+\\w[\\w-]+"

case class User(                                     // Scala
  names: List[String] :|
    Length[GreaterEqual[2]] & ForAll[Match[nameRegex]],
  age: Int :| Positive0,
  email: String :| Match[emailRegex])

final class Match[V <: String]
object Match:
  inline given [V <: String]: Constraint[String, Match[V]] with
    inline def pattern = constValue[V]
    ...
```

\* Используется библиотека Iron, <https://github.com/Iltotore/iron>

Привет  
○

Ограничения в типах  
○○○○○○●○

Деривация  
○○○○

Внешний источник  
○

Заключение  
○○○

# Зависимые типы

# Зависимые типы

```
record User where
  constructor MkUser
  names          : List String

  age            : Nat
  email          : String
```

— Idris

## Зависимые типы

```
record User where
  constructor MkUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}

  age            : Nat
  email          : String
```

— Idris



## Зависимые типы

```
nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]".erx
```

```
record User where
  constructor MkUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}

  age            : Nat
  email          : String
```

— Idris

## Зависимые типы

```
nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]".erx
```

```
record User where                                     — Idris
  constructor MkUser
  names      : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age         : Nat
  email       : String
```

## Зависимые типы

```
nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]".erx
emailRegex = #"[\\w·-]+@(?:[\\w-]+\\.)+\\w[\\w-]+"#.erx
```

```
record User where                                     — Idris
  constructor MkUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age            : Nat
  email          : String
```

## Зависимые типы

```
nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]".erx
emailRegex = #"[\\w·-]+@(?:[\\w-]+\\.)+\\w[\\w-]+"#.erx
```

```
record User where                                     — Idris
  constructor MkUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}
```

## Зависимые типы

```
nameRegex  = "[A-Z][A-Za-z-]*[A-Za-z]".erx
emailRegex = #"[\\w·-]+@(?:[\\w-]+\\.)+\\w[\\w-]+"#.erx
```

```
record User where                                     — Idris
  constructor MkUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}
```

```
good = MkUser ["Denis", "Buzdalov"] 35 "test@example.com"
```

## Зависимые типы

```
nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]".erx
emailRegex = #"[\\w·-]+@(?:[\\w-]+\\.)+\\w[\\w-]+"#.erx
```

```
record User where                                     — Idris
  constructor MkUser
  names      : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age         : Nat
  email       : String
  {auto evEmailOk : MatchesWhole emailRegex email}

failing "Can't find"
  bad = MkUser ["Denis", "Buzdalov"] 35 "bad@email"
```

## Зависимые типы

```
nameRegex  = "[A-Z][A-Za-z-]*[A-Za-z]".erx
emailRegex = #"[\\w·-]+@(?:[\\w-]+\\.)+\\w[\\w-]+"#.erx
```

```
record User where                                     — Idris
  constructor MkUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}

failing "Can't find"
  bad = MkUser ["Denis"] 35 "test@example.com"
```

## Но это же то же самое, что refined-типы?

```
nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]".erx
emailRegex = #"[\\w·-]+@(?:[\\w-]+\\.)+\\w[\\w-]+"#.erx
```

```
record User where                                     — Idris
  constructor MkUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}

failing "Can't find"
  bad = MkUser ["Denis"] 35 "test@example.com"
```



Но это же то же самое, что refined-типы? Нет!

```
nameRegex = "[A-Z][A-Za-z-]*[A-Za-z]".erx
emailRegex = #"[\\w·-]+@(?:[\\w-]+\\.)+\\w[\\w-]+"#.erx
```

```
record User where                                     — Idris
  constructor MkUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}

failing "Can't find"
  bad = MkUser ["Denis"] 35 "test@example.com"
```

Но это же то же самое, что refined-типы? Нет!

```
length : List a → Nat
(≥) : Ord a ⇒ a → a → Bool
```

```
record User where                                     — Idris
  constructor MkUser
  names      : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age         : Nat
  email       : String
  {auto evEmailOk : MatchesWhole emailRegex email}
```

```
failing "Can't find"
  bad = MkUser ["Denis"] 35 "test@example.com"
```

# Но это же то же самое, что refined-типы? Нет!

```
data So : Bool → Type where
  Oh : So True
```

```
record User where                                     — Idris
  constructor MkUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}
```

```
failing "Can't find"
  bad = MkUser ["Denis"] 35 "test@example.com"
```

## Но это же то же самое, что refined-типы? Нет!

```
MatchesWhole : RegExp a → String → Type
MatchesWhole r s = IsJust (matchesWhole r s)
```

```
record User where                                     — Idris
  constructor MkUser
  names      : List String
  {auto evLenOk   : So (length names ≥ 2)}
  {auto evNameOk  : All (MatchesWhole nameRegex) names}
  age         : Nat
  email       : String
  {auto evEmailOk : MatchesWhole emailRegex email}
```

```
failing "Can't find"
  bad = MkUser ["Denis"] 35 "test@example.com"
```

## Прелести зависимых типов

```
record SimpleUser where                                     — Idris
  constructor MkSimpleUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}
```

## Прелести зависимых типов

```
record SimpleUser where                                     — Idris
  constructor MkSimpleUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}

total firstName : SimpleUser → String
firstName (MkSimpleUser {names = first :: _} {}) = first
```

## Прелести зависимых типов

```
emailRegex = #"[\\w\\.-]+@((?:[\\w\\.-]+\\.)+\\w[\\w\\.-]+)"#.erx
```

```
record SimpleUser where                                     — Idris
  constructor MkSimpleUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}
```

```
total firstName : SimpleUser → String
firstName (MkSimpleUser {names = first :: _} {}) = first
```

## Прелести зависимых типов

```
emailRegex : RegExp (Vect 1 String)
emailRegex = #"[\\w·-]+@((?:[\\w-]+\\.)+\\w[\\w-]+)"#.erx
```

```
record SimpleUser where                                     — Idris
  constructor MkSimpleUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}
```

```
total firstName : SimpleUser → String
firstName (MkSimpleUser {names = first :: _} {}) = first
```



## Прелести зависимых типов

```
emailRegex : RegExp (Vect 1 String)
emailRegex = #"[\\w·-]+@((?:[\\w-]+\\.)+\\w[\\w-]+)"#.erx
```

```
record SimpleUser where                                     — Idris
  constructor MkSimpleUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRegex email}
```

```
total domainName : SimpleUser → String
domainName (MkSimpleUser {email} {}) =
  matchWholeResult emailRegex email ▷
    \[domain] ⇒ domain          — ← Vect 1 String → String
```

## Прелести зависимых типов

```
record ParamUser (emailRx : RegExp (Vect 1 String)) where
  constructor MkParamUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRx email}
```

```
total domainName : SimpleUser → String
domainName (MkSimpleUser {email} {}) =
  matchWholeResult emailRegex email ▷
    \[domain] ⇒ domain           — ← Vect 1 String → String
```

## Прелести зависимых типов

```
record ParamUser (emailRx : RegExp (Vect 1 String)) where
  constructor MkParamUser
  names          : List String
  {auto evLenOk   : So (length names ≥ 2)}
  age            : Nat
  email          : String
  {auto evEmailOk : MatchesWhole emailRx email}

total domainName : ParamUser emailRegex → String
domainName (MkParamUser {email} {}) =
  matchWholeResult emailRegex email ▷
    \[domain] ⇒ domain           — ← Vect 1 String → String
```

Привет  
○

Ограничения в типах  
○○○○○○○○

Деривация  
●○○○

Внешний источник  
○

Заключение  
○○○

```
schema = #"""
{ "type": "object", "properties": {
  "names": { "type": "array",
    "minItems": 2,
    "items": { "type": "string",
      "pattern": "[A-Z][A-Za-z\\-]*[A-Za-z]" }},
  "age": { "type": "integer", "minimum": 0 },
  "email": { "type": "string",
    "pattern": "[\\w\\.-]+@(?:[\\w-]+\\.)+\\w[\\w-]+" }}}
"""#
```

— Idris

schema = #""— Idris

```
{ "type": "object", "properties": {  
  "names": { "type": "array",  
    "minItems": 2,  
    "items": { "type": "string",  
      "pattern": "[A-Z][A-Za-z\\-]*[A-Za-z]" }},  
  "age": { "type": "integer", "minimum": 0 },  
  "email": { "type": "string",  
    "pattern": "[\\w\\.-]+@[?:(\\w-+\\.)+\\w[\\w-]+" }}}
```

""#

User : Type

User = %runElab deriveJsonSchema (Str schema)

```
schema = #""""                                     — Idris
  { "type": "object", "properties": {
    "names": { "type": "array",
      "minItems": 2,
      "items": { "type": "string",
        "pattern": "[A-Z][A-Za-z\\-]*[A-Za-z]" }},
    "age": { "type": "integer", "minimum": 0 },
    "email": { "type": "string",
      "pattern": "[\\w\\.-]+@[?:[\\w-]+\\.)+\\w[\\w-]+" }}}
```

```
User : Type
```

```
User = %runElab deriveJsonSchema (Str schema)
```

```
good, bad : User
```

```
good = MkUser ["Denis", "Buzdalov"] 35 "test@example.com"
```

```
schema = #""
  { "type": "object", "properties": {
    "names": { "type": "array",
      "minItems": 2,
      "items": { "type": "string",
        "pattern": "[A-Z][A-Za-z\\-]*[A-Za-z]" }},
    "age": { "type": "integer", "minimum": 0 },
    "email": { "type": "string",
      "pattern": "[\\w\\.-]+@(?:[\\w-]+\\.)+\\w[\\w-]+" }}}
""#
```

— Idris

```
User : Type
```

```
User = %runElab deriveJsonSchema (Str schema)
```

```
good, bad : User
```

```
good = MkUser ["Denis", "Buzdalov"] 35 "test@example.com"
```

```
failing "Can't find"
```

```
bad = MkUser ["Denis", "Buzdalov"] 35 "bad@email"
```



```
altSchema = #""
{ "type": "object", "properties": {
  "name": { "type": "string",
    "pattern": "(?:[A-Z][A-Za-z\\-]* )+[A-Z][A-Za-z\\-]*" },
  "age": { "type": "object", "properties": {
    "value": { "type": "integer",
      "minimum": 0, "maximum": 150 }}}}}
""#
```

— Idris

```
altSchema = #""""                                     — Idris
{ "type": "object", "properties": {
  "name": { "type": "string",
    "pattern": "(?:[A-Z][A-Za-z\\-]* )+[A-Z][A-Za-z\\-]*" },
  "age": { "type": "object", "properties": {
    "value": { "type": "integer",
      "minimum": 0, "maximum": 150 }}}}}
""""#
```

AltUser : Type

AltUser = %runElab deriveJsonSchema (Str altSchema)

```
altSchema = #""""                                     — Idris
{ "type": "object", "properties": {
  "name": { "type": "string",
    "pattern": "(?:[A-Z][A-Za-z\\-]* )+[A-Z][A-Za-z\\-]*" },
  "age": { "type": "object", "properties": {
    "value": { "type": "integer",
      "minimum": 0, "maximum": 150 }}}}}
""""#
```

AltUser : Type

AltUser = %runElab deriveJsonSchema (Str altSchema)

altUser = MkAltUser "Denis Buzdalov" (MkAge 35)

```
altSchema = #""                                     — Idris
{ "type": "object", "properties": {
  "name": { "type": "string",
    "pattern": "(?:[A-Z][A-Za-z\\-]* )+[A-Z][A-Za-z\\-]*" },
  "age": { "type": "object", "properties": {
    "value": { "type": "integer",
      "minimum": 0, "maximum": 150 }}}}}
""#
```

```
AltUser : Type
```

```
AltUser = %runElab deriveJsonSchema (Str altSchema)
```

```
someAge : AgeTy      ;      someName : String
```

```
someAge = MkAge 35    ;      someName = "Denis Buzdalov"
```

```
altUser = MkAltUser "Denis Buzdalov" (MkAge 35)
```

```
altSchema = #""                                     — Idris
{ "type": "object", "properties": {
  "name": { "type": "string",
    "pattern": "(?:[A-Z][A-Za-z\\-]* )+[A-Z][A-Za-z\\-]*" },
  "age": { "type": "object", "properties": {
    "value": { "type": "integer",
      "minimum": 0, "maximum": 150 }}}}}
""#
```

```
AltUser : Type
```

```
AltUser = %runElab deriveJsonSchema (Str altSchema)
```

```
someAge : AgeTy      ;      someName : String
```

```
someAge = MkAge 35    ;      someName = "Denis Buzdalov"
```

```
altUser = MkAltUser someName someAge
```

```
altSchema = #""                                     — Idris
{ "type": "object", "properties": {
  "name": { "type": "string",
    "pattern": "(?:[A-Z][A-Za-z\\-]* )+[A-Z][A-Za-z\\-]*" },
  "age": { "type": "object", "properties": {
    "value": { "type": "integer",
      "minimum": 0, "maximum": 150 }}}}}
""#
```

```
AltUser : Type
```

```
AltUser = %runElab deriveJsonSchema (Str altSchema)
```

```
someAge : AgeTy      ;      someName : String
```

```
someAge = MkAge 35    ;      someName = "Denis Buzdalov"
```

```
altUser = MkAltUser someName someAge
```

```
ageOk = (altUser.age.value = 35) `the` Refl
```

Привет  
○

Ограничения в типах  
○○○○○○○○

Деривация  
○○●○

Внешний источник  
○

Заключение  
○○○

Но чёрт возьми... Как?

## Но чёрт возьми... Как?

```
interface Monad m => Elaboration m where
```



## Но чёрт возьми... Как?

```
interface Monad m => Elaboration m where  
  check      : TImp -> m expected
```

## Но чёрт возьми... Как?

```
interface Monad m => Elaboration m where
  check      : TTImp → m expected
  quote     : val → m TTImp
```

## Но чёрт возьми... Как?

```
interface Monad m => Elaboration m where
  check      : TImp → m expected
  quote      : val → m TImp
  declare    : List Decl → m ()
```

## Но чёрт возьми... Как?

```
interface Monad m => Elaboration m where
  check      : TImp → m expected
  quote      : val → m TImp
  declare    : List Decl → m ()
  getCurrentFn : m (SnocList Name)
  ...
```

## Но чёрт возьми... Как?

```
interface Monad m => Elaboration m where
  check      : TImp → m expected
  quote      : val → m TImp
  declare    : List Decl → m ()
  getCurrentFn : m (SnocList Name)
  ...
```

```
loadJsonSchema : SchemaSource → Elab JsonSchema
```

## Но чёрт возьми... Как?

```
interface Monad m => Elaboration m where
  check      : TTImp → m expected
  quote      : val → m TTImp
  declare    : List Decl → m ()
  getCurrentFn : m (SnocList Name)
  ...
```

```
loadJsonSchema : SchemaSource → Elab JsonSchema
deriveToplevelJsonType : Name → JsonSchema → (List Decl, TTImp)
```

## Но чёрт возьми... Как?

```
interface Monad m => Elaboration m where
  check      : TImp → m expected
  quote      : val → m TImp
  declare    : List Decl → m ()
  getCurrentFn : m (SnocList Name)
  ...
```

```
loadJsonSchema : SchemaSource → Elab JsonSchema
deriveToplevelJsonType : Name → JsonSchema → (List Decl, TImp)
```

```
deriveJsonSchema : SchemaSource → Elab Type
deriveJsonSchema shs = do
  n ← prepareTyName <$> getCurrentFn
  js ← loadJsonSchema shs
  (decls, ty) ← deriveToplevelJsonType n js
  declare decls
  check ty
```

Привет  
○

Ограничения в типах  
○○○○○○○○

Деривация  
○○●

Внешний источник  
○

Заключение  
○○

Хотите подробностей?



## Хотите подробностей?

```
record DerivationResult where
  constructor DR
  decls : List Decl; constrs : List (Name → TTImp); type : TTImp

deriveJsonType : Name → JsonSchema → DerivationResult
```

## Хотите подробностей?

```
record DerivationResult where
  constructor DR
  decls : List Decl; constrs : List (Name → TTImp); type : TTImp

deriveJsonType : Name → JsonSchema → DerivationResult
...
deriveJsonType n (SArray items minItems maxItems) = do
  let DR subds subcs subty = deriveJsonType n items
  let subcs = subcs <&> \c, n ⇒ let m = UN (Basic "^x^")
    `(All ~(lam MW ExplicitArg (Just m) subty (c m)) ~(var n))
  let llen = minItems <&> \mi, n ⇒
    `(fromInteger ~(primVal (BI mi)) `Data.Nat.LTE` length ~(var n))
  let rlen = maxItems <&> \ma, n ⇒
    `(length ~(var n) `Data.Nat.LTE` fromInteger ~(primVal (BI ma)))
  DR subds (subcs ++ mapMaybe id [llen, rlen]) `(List ~subty)
```

## Хотите подробностей?

```
record DerivationResult where
  constructor DR
  decls : List Decl; constrs : List (Name → TTImp); type : TTImp

deriveJsonType : Name → JsonSchema → DerivationResult
...
deriveJsonType n (SArray items minItems maxItems) = do
  let DR subds subcs subty = deriveJsonType n items
  let subcs = subcs <&> \c, n => let m = UN (Basic "^x^")
    `(All ~(lam MW ExplicitArg (Just m) subty (c m)) ~(var n))
  let llen = minItems <&> \mi, n =>
    `(fromInteger ~(primVal (BI mi)) `Data.Nat.LTE` length ~(var n))
  let rlen = maxItems <&> \ma, n =>
    `(length ~(var n) `Data.Nat.LTE` fromInteger ~(primVal (BI ma)))
  DR subds (subcs ++ mapMaybe id [llen, rlen]) `(List ~subty)
deriveJsonType _ (SString pat) = do
  let patConstraint = \n =>
    `(MatchesWhole ~(primVal (Str pat)).erx ~(var n))
  DR empty [patConstraint] (primVal (PrT StringType))
...
```

Привет  
○

Ограничения в типах  
○○○○○○○○

Деривация  
○○○○

Внешний источник  
●

Заключение  
○○○

Truly single source of truth

# Truly single source of truth

```
interface Monad m => Elaboration m where
  check      : TTImp → m expected
  quote      : val → m TTImp
  declare    : List Decl → m ()
  getCurrentFn : m (SnocList Name)
  ...
```

# Truly single source of truth

```
interface Monad m => Elaboration m where
  check      : TImp → m expected
  quote      : val → m TImp
  declare    : List Decl → m ()
  getCurrentFn : m (SnocList Name)
  ...
  readFile : LookupDir → (path : String) → m (Maybe String)
  ...
```

# Truly single source of truth

```
interface Monad m => Elaboration m where
  check      : TImp → m expected
  quote      : val → m TImp
  declare    : List Decl → m ()
  getCurrentFn : m (SnocList Name)
  ...
  readFile   : LookupDir → (path : String) → m (Maybe String)
  ...

loadJsonSchema : SchemaSource → Elab JsonSchema
```

## Truly single source of truth

```
interface Monad m => Elaboration m where
  check      : TImp → m expected
  quote      : val → m TImp
  declare    : List Decl → m ()
  getCurrentFn : m (SnocList Name)
  ...
  readFile   : LookupDir → (path : String) → m (Maybe String)
  ...

loadJsonSchema : SchemaSource → Elab JsonSchema

User = %runElab deriveJsonSchema (File "user.json")
```



## Truly single source of truth

```
interface Monad m => Elaboration m where
  check      : TImp → m expected
  quote      : val → m TImp
  declare    : List Decl → m ()
  getCurrentFn : m (SnocList Name)
  ...
  readFile    : LookupDir → (path : String) → m (Maybe String)
  ...

loadJsonSchema : SchemaSource → Elab JsonSchema

User = %runElab deriveJsonSchema (File "user.json")

good = MkUser ["Denis", "Buzdalov"] 35 "test@example.com"
```

## Truly single source of truth

```
interface Monad m => Elaboration m where
  check      : TImp → m expected
  quote      : val → m TImp
  declare    : List Decl → m ()
  getCurrentFn : m (SnocList Name)
  ...
  readFile   : LookupDir → (path : String) → m (Maybe String)
  ...
```

```
loadJsonSchema : SchemaSource → Elab JsonSchema
```

```
User = %runElab deriveJsonSchema (File "user.json")
```

```
good = MkUser ["Denis", "Buzdalov"] 35 "test@example.com"
failing "Can't find"
  bad = MkUser ["Denis", "Buzdalov"] 35 "bad@email"
```

Привет  
○

Ограничения в типах  
○○○○○○○○

Деривация  
○○○○

Внешний источник  
○

Заключение  
●○○

# В сухом остатке

## В сухом остатке

- Зависимые типы — круто

## В сухом остатке

- Зависимые типы — круто
- Метапрограммирование в специальной монаде — мощно

## В сухом остатке

- Зависимые типы — круто
- Метапрограммирование в специальной монаде — мощно
- Когда эта монада суперкрутая — ещё мощнее

## В сухом остатке

- Зависимые типы — круто
- Метапрограммирование в специальной монаде — мощно
- Когда эта монада суперкрутая — ещё мощнее
- Скаловские `inline`'ы + система макросов весьма мощна

## В сухом остатке

- Зависимые типы — круто
- Метапрограммирование в специальной монаде — мощно
- Когда эта монада суперкрутая — ещё мощнее
- Скаловские `inline`'ы + система макросов весьма мощна
- Если привлечь ещё сложных фич — ещё мощнее



## В сухом остатке

- Зависимые типы — круто
- Метапрограммирование в специальной монаде — мощно
- Когда эта монада суперкрутая — ещё мощнее
- Скаловские `inline`'ы + система макросов весьма мощна
- Если привлечь ещё сложных фич — ещё мощнее
- Но зависимые типы — круче\* ;-)

\* очень субъективное мнение

# Поиграться дома



Эта презентация



Код с презентации

## Мнение про доклад

