

Idris

2014 г.

Idris

- ▶ Haskell-подобный,
- ▶ с зависимыми типами,
- ▶ строгий по-умолчанию,
- ▶ с опциональной проверкой на тотальность,
- ▶ с тактиками,
- ▶ ...

Haskell-подобный

```
data MyList a = Nil | (::) a (MyList a)
```

```
(++) : MyList a → MyList a → MyList a
```

```
[] ++ ys = ys
```

```
(x :: xs) ++ ys = x :: (xs ++ ys)
```

```
instance Functor MyList where
```

```
  map f Nil = Nil
```

```
  map f (x :: xs) = f x :: map f xs
```

Haskell-подобный

instance *Applicative* *MyList* **where**

pure *x* = [*x*]

[] <\$> _ = []

(*f* :: *fs*) <\$> *xs* = *map* *f* *xs* ++ (*fs* <\$> *xs*)

instance *Monad* *MyList* **where**

[] »= _ = []

(*x* :: *xs*) »= *f* = *f* *x* ++ (*xs* »= *f*)

test : *MyList* *Int*

test = **do**

f ← [*id*, (*2)]

x ← [3, 4]

return \$ *f* *x*

С зависимыми типами

data *MyVect* : *Nat* \rightarrow (*a* : *Type*) \rightarrow *Type* **where**

Nil : *MyVect* 0 *a*

(::) : *a* \rightarrow *MyVect* *n* *a* \rightarrow *MyVect* (*S* *n*) *a*

(++) : *MyVect* *n* *a* \rightarrow *MyVect* *m* *a* \rightarrow *MyVect* (*n* + *m*) *a*

[] ++ *ys* = *ys*

(*x* :: *xs*) ++ *ys* = *x* :: (*xs* ++ *ys*)

infix 9 !!

(!!) : *MyVect* *n* *a* \rightarrow *Fin* *n* \rightarrow *a*

(*x* :: *xs*) !! *fZ* = *x*

(*x* :: *xs*) !! (*fS* *y*) = *xs* !! *y*

Строгий по-умолчанию

broken : *Int* → *Int*

broken 0 = 1

broken *n* = *n* * *broken* (*n* − 1)

ifThenElse : *Bool* → *a* → *a* → *a*

ifThenElse *True* *t* *_* = *t*

ifThenElse *False* *_* *f* = *f*

> *ifThenElse* *True* 0 (*broken* (−1))

Интерпретатор:

0 : *Int*

Скомпилированный код(с точностью до оптимизаций):

segmentation fault ./a.out

С опциональной проверкой на тотальность

total *myHead* : *List a* \rightarrow *a*

myHead (*x* :: *xs*) = *x*

> Main.myHead is not total as there are missing cases

%default *total*

go : *Int*

go = *go*

> Main.go is possibly not total due to recursive path Main.go

С тактиками

```
lemma_applicative_identity : (vs : MyList a) → (pure id <$> vs = vs)
lemma_applicative_identity [] = refl
lemma_applicative_identity (v :: vs) =
  let rec = lemma_applicative_identity vs
  in ?lemma_applicative_identity_rhs

lemma_applicative_identity_rhs = proof
  intro a, x, xs, rec
  rewrite rec
  trivial
```


- Effects вместо трансформеров

$$f : \{ [STDIO, STATE Int] \} Eff ()$$

- Именованные инстансы

```
instance [myord] Ord Int where
```

...

```
sort @{myord} [2, 1, 3]
```

- Idiom brackets(для аппликативных функторов)

$$f : Maybe Int \rightarrow Maybe Int \rightarrow Maybe Int$$

$$f \times y = [|x + y|]$$

- ...
▶ !-нотация(для монад)

$$f : \text{Maybe } \text{Bool} \rightarrow \text{Maybe } a \rightarrow \text{Maybe } a \rightarrow \text{Maybe } a$$
$$f \times t \ f = \text{if } !x \text{ then } t \text{ else } f$$

- ▶ Опциональная ленивость

$$\text{data Lazy} : \text{Type} \rightarrow \text{Type} \text{ where}$$
$$\text{Delay} : a \rightarrow \text{Lazy } a$$
$$\text{Force} : \text{Lazy } a \rightarrow a$$

- ▶ Изменяемый синтаксис

$$\text{syntax if [test] then [t] else [e] = boolElim test (Delay t) (Delay e)$$

- ▶ Минимальный вывод типов в **where**
- ▶ Гетерогенное равенство

data $(=) : a \rightarrow b \rightarrow \text{Type}$ **where**
 $\text{refl} : x = x$

- ▶ Type providers TODO
- ▶ FFI with C TODO

- ▶ **public, abstract, private** спецификаторы видимости

```
%access public
abstract f : Int → Int
```

- ▶ **records**

```
record R : Type where
  MkR : (f1 : Int) → (f2 : String) → R
```

- ▶ Levels are implicit TODO
- ▶ auto implicit args TODO

TODOs

- ▶ Proof automation
- ▶ More better termination checker
- ▶ More better editor support (goto definition, autocomplete, ...)
- ▶ More bindings (incl. low-level C bindings)
- ▶ More backends (e.g. GHC)
- ▶ Bugfixing