Idris

2014 г.

Idris

- Haskellоподобный,
- с зависимыми типами,
- строгий по-умолчанию,
- с опциональной проверкой на тотальность,
- с тактиками,
- . . .

Haskellоподобный

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*

Haskellоподобный

```
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 → (a : Type) → Type where
  Nil : MyVect 0 a
  (::) : a → MyVect n a → MyVect (S n) a


(++) : MyVect n a → MyVect m a → MyVect (n + m) a
[] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)


infix 9 !!


(!!) : MyVect n a → Fin n → 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 → 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)

  $f : Maybe\ Int \to Maybe\ Int \to Maybe\ Int$
  $f\ x\ y = [|x + y|]$

- !-нотация(для монад)

  $f : Maybe\ Bool \rightarrow Maybe\ a \rightarrow Maybe\ a \rightarrow Maybe\ a$

  $f\ x\ t\ f =$ **if** $!x$ **then** t **else** f

- Опциональная ленивость

  data $Lazy : Type \rightarrow Type$ **where**
     $Delay : a \rightarrow Lazy\ a$

     $Force : Lazy\ a \rightarrow a$

- Изменяемый синтаксис

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

. . .

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

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

- Type providers TODO
- FFI with C TODO

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

  %**access** *public*
  abstract $f : Int \rightarrow Int$

- **record**s

  record R : *Type* **where**
  $MkR : (f1 : Int) \rightarrow (f2 : String) \rightarrow R$

- Levels are implicit
- auto implicit args

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