Экстракция кода из Agda в Haskell

Шабалин Александр

научный руководитель

доц. Москвин Д. Н.

Академический университет

2013 г.

Мотивирующий пример

TODO: Пример, который легко ломается в хаскеле и легко чинится зависимыми типами.

Зависимые типы

System F:

$$\mathbf{Term} ::= \mathbf{Var} \mid \lambda x.\ \mathbf{Term}(x) \mid \mathbf{Term}\ \mathbf{Term}$$
$$\mathbf{Type} ::= \mathbf{TVar} \mid \mathbf{Type} \to \mathbf{Type} \mid \forall x.\ \mathbf{Type}(x)$$

$$\mathbf{\Gamma} \vdash \mathbf{Term} : \mathbf{Type}, \quad \mathbf{\Gamma} = \mathbf{Var} : \mathbf{Type}, \ldots$$

Зависимые типы

System F:

$$\mathbf{Term} ::= \mathbf{Var} \mid \lambda x.\ \mathbf{Term}(x) \mid \mathbf{Term}\ \mathbf{Term}$$
$$\mathbf{Type} ::= \mathbf{TVar} \mid \mathbf{Type} \to \mathbf{Type} \mid \forall x.\ \mathbf{Type}(x)$$

$$\mathbf{\Gamma} \vdash \mathbf{Term} : \mathbf{Type}, \quad \mathbf{\Gamma} = \mathbf{Var} : \mathbf{Type}, \ldots$$

Зависимые типы:

$$\begin{aligned}\mathbf{Term} ::= {} & \mathbf{Var} \\ & \mid \mathbf{Term}\ \mathbf{Term} \\ & \mid \lambda x.\ \mathbf{Term}(x) \mid (x : \mathbf{Term}) \to \mathbf{Term}(x) \\ & \mid (\mathbf{Term}, \mathbf{Term}) \mid (x : \mathbf{Term}) \times \mathbf{Term}(x) \\ & \mid \mathbf{Set}\end{aligned}$$

$$\mathbf{\Gamma} \vdash \mathbf{Term} : \mathbf{Term}, \quad \mathbf{\Gamma} = \mathbf{Var} : \mathbf{Term}, \ldots$$

Agda

Язык с зависимыми типами и синтаксисом, похожим на Haskell.

Agda - примеры

data *работает аналогично GADT в Haskell*

```
data List (A : Set) : Set where
  <> : List A
  _::_ : A → List A → List A
```

{ . . . } — *необязательный аргумент(компилятор сам подставит)*

```
list-length : {A : Set} → List A → Nat
list-length <> = 0
list-length (x :: xs) = list-length xs + 1
```

Agda - примеры

```
data Vec (A : Set) : Nat → Set where
  nil : Vec A 0
  cons : {n : Nat} → A → Vec A n → Vec A (n + 1)

list-to-vec : {A : Set} → (xs : List A) → Vec A (list-length xs)
list-to-vec <> = nil
list-to-vec (x :: xs) = cons x (list-to-vec xs)
```

Agda - примеры

$\{a_1\ a_2 : A\}(b : B) \to C$ — *синтаксический сахар для*

$\{a_1 : A\} \to \{a_2 : A\} \to (b : B) \to C$

$$\text{zip-vec} : \{A\ B : Set\}\{n : Nat\} \to Vec\ A\ n \to Vec\ B\ n \to Vec\ (A \times B)\ n$$

$$\text{zip-vec nil nil} = \text{nil}$$

$$\text{zip-vec (cons } x\ xs\text{) (cons } y\ ys\text{)} = \text{cons } (x, y)\ \text{(zip-vec } xs\ ys\text{)}$$

$$\text{zip-vec nil (cons } y\ ys\text{)} = \ldots$$

$$\text{zip-vec (cons } x\ xs\text{) nil} = \ldots$$

эти 2 клоза даже написать нельзя - типы не сойдутся

Agda

TODO: Пример с первого слайда

Компилятор MAlonzo

$$\textbf{data}\ \mathit{Vec}\ (A : \mathit{Set}) : \mathit{Nat} \to \mathit{Set}\ \textbf{where}$$
$$\quad \mathrm{nil} : \mathit{Vec}\ A\ 0$$
$$\quad \mathrm{cons} : \{n : \mathit{Nat}\} \to A \to \mathit{Vec}\ A\ n \to \mathit{Vec}\ A\ (n + 1)$$
$$\text{vec-map} : \{A\ B : \mathit{Set}\}\{n : \mathit{Nat}\} \to (A \to B) \to \mathit{Vec}\ A\ n \to \mathit{Vec}\ B\ n$$
$$\text{vec-map}\ f\ \mathrm{nil} = \mathrm{nil}$$
$$\text{vec-map}\ f\ (\mathrm{cons}\ x\ xs) = \mathrm{cons}\ (f\ x)\ (\text{vec-map}\ f\ xs)$$

$$\textbf{data}\ T_1\ a_0\ a_1\ a_2 = C_2 \mid C_3\ a_0\ a_1\ a_2$$
$$\mathrm{d}_4\ v_0\ v_1\ v_2\ v_3\ (C_2) = \mathrm{cast}\ C_2$$
$$\mathrm{d}_4\ v_0\ v_1\ v_2\ v_3\ (C_3\ v_4\ v_5\ v_6) = \mathrm{cast}\ (C_3\ (\mathrm{cast}\ v_4)\ (\mathrm{cast}\ (v_3\ (\mathrm{cast}\ v_5))))$$
$$\quad (\mathrm{cast}\ (\mathrm{d}_4\ (\mathrm{cast}\ v_0)\ (\mathrm{cast}\ v_1)\ (\mathrm{cast}\ v_4)\ (\mathrm{cast}\ v_3)\ (\mathrm{cast}\ v_6))))$$
$$\textbf{ghci>}\ \textbf{:t}\ \mathrm{d}_4$$
$$\mathrm{d}_4\ ::\ a \to a_1 \to a_2 \to a_3 \to (T_1\ t\ t_1\ t_2) \to b$$

Coq

```
Inductive vec (A : Set) : nat → Set := nil : vec A 0
 | cons : ∀n : nat, A → vec A n → vec A (n + 1).
Fixpoint vMap (A : Set) (B : Set) n (f : A → B)
   (v : vec A n) : vec B n :=
      match v with nil ⇒ nil _
      | cons _ x xs ⇒ cons _ _ (f x) (vMap _ _ _ f xs)
end.
```

```
data Vec a = Nil | Cons Nat a (Vec a)
vMap :: Nat → (a1 → a2) → Vec a1 → Vec a2
vMap n f v = case v of
   Nil → Nil
   Cons n0 x xs → Cons n0 (f x) (vMap n0 f xs)
```

Что хочется

Идея: Экспортировать только то, что при преобразовании типов не потеряет информацию(инварианты).

Сначала разрешаем только типы, которые имеют полный аналог в Haskell. Потом добавляем отдельные случаи зависимых типов, представляемых в Haskell.

Что сделано

С помощью прагм `{-# EXPORT agda-name haskell-name #-}` можно экспортировать типы и функции(тип которых выводится автоматически).
Но типы экспортируются только как абстрактные(`newtype`), а не с конструкторами(`data`).
TODO: go into detail here

А что можно попробовать

1. разрешить экспортировать типы с конструкторами
2. зависимые типы вроде $f : \{A\ B : Set\}\{n : Nat\} \to Vec\ A\ n \to Vec\ B\ n \to Vec\ (A \times B)\ n$ могут быть экспортированы, если n не используется внутри f
3. есть трюк, позволяющий экспортировать типы вроде $f : \{A : Set\}(n : Nat) \to A \to Vec\ a\ n$

Q&A