

# Applied Functional Programming USCS 2009

Andreas Resios, Călin Juravle, Ruud Koot, Cristian Tălău

Department of Information and Computing Sciences
Utrecht University

August 27, 2009

#### **B1.** The Lambda Calculus Interpreter

## **Approach**

- ▶ We developed an interpreter along the lines of GHCi.
- ▶ We started with the untyped lambda calculus and implemented types and basic type checking.

4日 > 4間 > 4 国 > 4 国 > 国 >

#### Lambda calculus

- ▶ Introduced by Church in 1936.
- Simple formal language based on variables, function abstraction and application.
- ▶ It is equivalent in computing power as a Turing machine.
- ▶ It provides the theoretical basis for functional programming languages such as Haskell.



#### **Parser**

We used parser combinators to implement the following grammar:

#### **Terms**

$$\begin{array}{cccc} \mathbf{e} ::= \mathbf{x} & \text{variables} \\ & | & \mathbf{e} \ \mathbf{e} & \text{application} \\ & | & \lambda(\mathbf{x} : \tau) \ . \ \mathbf{e} & \text{lambda abstraction} \end{array}$$

$$\begin{array}{cccc} \tau ::= \alpha & \text{type variables} \\ & \tau \to \tau & \text{function type} \end{array}$$

#### **Abstract syntax**

◆□▶◆御▶◆団▶◆団▶ 団 めの◎

### **Catamorphism**

```
type LambdaAlgebra a r =
 \begin{array}{c} (\mathsf{a} \to \mathsf{r} \\ , \mathsf{r} \to \mathsf{r} \to \mathsf{r} \\ , \mathsf{Var} \to \mathsf{Type} \to \mathsf{r} \to \mathsf{r} \end{array} 
   foldExpr :: LambdaAlgebra a r \rightarrow LambdaExpr a \rightarrow r
  \begin{array}{ll} \text{foldExpr } q@(v,a,l) \; (\text{Var} \quad n \;\;) = v \; n \\ \text{foldExpr } q@(v,a,l) \; (\text{Appl e f}) = a \; (\text{foldExpr q e}) \; (\text{foldExpr q f}) \\ \text{foldExpr } q@(v,a,l) \; (\text{Lambda n t e}) = l \; n \; t \; (\text{foldExpr q e}) \end{array}
```



◆□▶◆圖▶◆臺▶◆臺▶ 臺 釣९@

## Sample algebra



## Sample algebra

```
\begin{array}{l} \text{freeAlgebra} :: \mathsf{NamedAlgebra} \; [\mathsf{Var}] \\ \text{freeAlgebra} = \\ & (\lambda \mathsf{x} \to [\mathsf{x}] \\ & , \mathsf{union} \\ & , \lambda \mathsf{a} \mathrel{\_} \mathsf{b} \to \mathsf{b} \setminus \setminus [\mathsf{a}] \\ & ) \\ \text{free} = \mathsf{foldExpr} \; \mathsf{freeAlgebra} \end{array}
```

4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶</p

#### **Implementation details**

- ► Evaluation strategy
- $ightharpoonup \alpha$ -conversion





[Faculty of Science

## **De Bruijn indices**

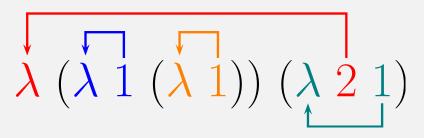


Figure: Kaustuv Chaudhuri

#### Another algebra

```
toDeBruijnAlgebra :: NamedAlgebra (Env \rightarrow DeBruijnExpr)
\begin{split} to De Bruijn Algebra &= (v, a, I) \\ \textbf{where} \ v \ n \ env &= Var \ (maybe \ (Right \ n) \ (Left) \ (lookup \ n \ env)) \end{split}
       a e f env = Appl (e env) (f env)
       In t e env = Lambda n t (e updateenv) where
                     updateenv =
                         (\mathsf{n},0):(\mathsf{map}\;(\lambda(\mathsf{a},\mathsf{b})\to(\mathsf{a},\mathsf{b}+1))\;\mathsf{env})
```

#### Type checking

To typecheck an expression we use the following inference rules to construct a proof.

Variables

$$\frac{\mathbf{x} : \tau \in \Gamma}{\Gamma \vdash \mathbf{x} : \tau}$$

#### **Application**

$$\frac{\Gamma \vdash \mathsf{e}_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash \mathsf{e}_2 : \tau_1}{\Gamma \vdash (\mathsf{e}_1 \; \mathsf{e}_2) : \tau_2}$$

#### Abstraction



#### Interactive environment

```
interactive :: StateT (Map String Expr) IO ()
interactive = do s \leftarrow get
   liftIO $ putStr (show (keys s) # "> ")
   i \leftarrow liftIO getLine
   case id of
       : : \to \mathbf{case} \ \mathsf{ic} \ \mathsf{of}
          '=' \rightarrow let (n,s) = splitOn isSpace ie
             in modify (insert n (fullpars s))
        \_ 
ightarrow \operatorname{\mathsf{do}} e \leftarrow return \$ fulleval s i
           liftIO \$ putStr (show e ++ "\n")
          interactive
```



4日 > 4 個 > 4 豆 > 4 豆 > 豆 めの()

### **Examples**

Examples

