Implementing Pure Lambda Calculus in Haskell

Chad Reynolds

Lambda Calculus

- Three forms of lambda terms:

  <terms> t ::= <variable> x | <abstraction> λx.t | <application> t t'

- Calculation is done through beta-reduction and substitution - (λx.t)t' = [t'/x]t

Lambda Calculus

- Different reduction strategies:

$$\underline{(\lambda x.\lambda y.y)((\lambda x.xx)\lambda x.xx)} \rightsquigarrow \lambda y.y$$

$$(\lambda x.\lambda y.y)(\underline{(\lambda x.xx)\lambda x.xx}) \rightsquigarrow (\lambda x.\lambda y.y)((\lambda x.xx)\lambda x.xx)$$

- Leftmost reduction will always result in a normal form if a normal form exists

Variable Capture

- What happens when reducing a term with free variables?

  $$\underline{(\lambda x.\lambda y.x)y} \rightsquigarrow \lambda y.\mathbf{y}$$

- When what we really want is:

  $$\underline{(\lambda x.\lambda y.x)y} \rightsquigarrow \lambda y.\mathbf{z}$$

De Bruijn Indices

- Avoid the variable capture problem by having variables represented as numbers referencing their binder

- Our new syntax becomes:

<terms> t ::= <variable> 0,1,... | <abstraction> λ.t | <application> t t'

- λx.x becomes λ.0 and λx.y becomes λ.1

De Bruijn Indices

- Beta-reduction becomes:

  $(\lambda.t)t' = [t'/0]t$

- Substitution involves comparison of the variables and addition/subtraction as it drills down

  $\underline{(\lambda.\lambda.1)(\lambda.0)}$

  $= [(\lambda.0)/0](\lambda.1)$

  $= \lambda.([(\lambda.0)/1]1)$

  $= \lambda.\lambda 0$

C. Hankin. An Introduction to Lambda Calculi for Computer Scientists. Texts in computing. Kings College, 2004.

Demo

Questions?