Lambda Calculus

- *Lambda calculus* provides a *theoretical basis* for functional programming
- *Lambda calculus* is also valuable as a *notation* to distinguish between a function and the value of the function
- This allows us to give a *satisfactory explanation* of programming language phenomena

1

Syntax of Lambda Calculus

- For all v in the set of variables, v is a lambda expression
- For all f in the set of function symbols, f is a lambda expression
- If v is a variable and l is a lambda expression, then $\lambda v.l$ is a lambda expression
- If m and n are lambda expressions, the (m, n) is a lambda expression

2

Syntax of Lambda Calculus

- We call terms of the form $\lambda v.l$ *abstractions*; the lambda expression v is said to be the *formal parameter* of the abstraction and the lambda expression l is said to be the *body* of the abstraction
- Terms of the form $(m\ n)$ are called *applications*; the lambda expression m is in the *function* position and n is in the *argument* position

3

Examples

- $\lambda x.\ \lambda y.\ (x\ y)$
- $(\lambda x.\ x\ (f\ a))$
- $((\lambda x.\ b\ c)\ (f\ \lambda y.\ c))$

4

Free Variables

- If the lambda expression is a variable, say v, then $FV[v] = \{v\}$
- If the lambda expression is a function symbol, say f, then $FV[f] = \varnothing$
- If the lambda expression is of the form $\lambda v.b$, then $FV[\lambda v.b] = FV[b] \backslash \{v\}$
- If the lambda expression is of the form $l\ m$, then $FV[l\ m] = FV[l] \cup FV[m]$
- Lambda expression e is *closed* if $FV[e] = \varnothing$

5

Free Variables

- We define the abstraction that binds the variable occurrence v to be the *innermost* abstraction of the form $\lambda v.m$ containing the variable occurrence
- If this abstraction exists, it is unique, and we call the formal parameter v the *binding occurrence* of all the occurrences of v that become bound in $\lambda v.m$
- $\lambda x.\ (\lambda y.\ (\lambda x.\ x\ (x\ y))\ (\lambda z.\ f\ \ x\ z))\ (g\ x)$

6