CMSC 330: Organization of Programming Languages

Lambda Calculus Introduction

Introduction

- We've seen that several language conveniences aren't strictly necessary
  - Multi-argument functions: use currying or tuples
  - Loops: use recursion
  - Side-effects: we don't need them either
- **Goal: come up with a "core" language that's as small as possible and still Turing complete**
  - This will give a way of illustrating important language features and algorithms

Revised Rule for Application

$$\frac{A;\ E_1 \rightarrow (A',\ \lambda x.E) \qquad A;\ E_2 \rightarrow v \qquad A,\ A',\ x{:}v;\ E \rightarrow v'}{A;\ (E_1\ \ E_2) \rightarrow\ v'}$$

- To apply something to an argument:
  - Evaluate it to produce a closure
  - Evaluate the argument (call-by-value)
  - Evaluate the body of the closure, in
    - The current environment, extended with the closure's environment, extended with the binding for the parameter

Example

$$\frac{\begin{array}{l}\bullet;\ (\text{fun x = (fun y = + x y)}) \rightarrow \quad (\bullet,\ \lambda x.(\text{fun y = + x y})) \\ \qquad\qquad \bullet;\ 3 \rightarrow\ 3 \\ x{:}3;\ (\text{fun y = + x y}) \rightarrow \quad (x{:}3,\ \lambda y.(+\ x\ y))\end{array}}{\bullet;\ (\text{fun x = (fun y = + x y))}\ \ 3 \rightarrow \quad (x{:}3,\ \lambda y.(+\ x\ y))}$$

Lambda Calculus

- A lambda calculus expression is defined as

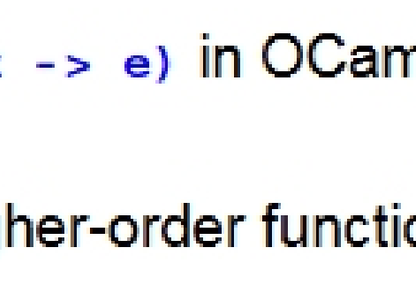
e ::= x	variable
| λx.e	function
| e e	function application

- λx.e is like `(fun x -> e)` in OCaml

- That's it! Only higher-order functions

Intuitive Understanding

- Before we work more with the mathematical notation of lambda calculus, we're going to play a puzzle game!



- From: http://worrydream.com/AlligatorEggs/