Everything is a Function

Chen Ding

Background

• Turing machine
  • model of computers
  • imperative instructions, re writable memory
  • sequential, implicit dependence
• Church’s Lambda calculus
  • model of abstraction
  • everything is a function
    • the language of mathematics
    • do everything by composing functions
    • no mutable state, no side effects
  • explicit value passing
LC Language

- Source
  - P. J. Landin 1964
  - Matthias Felleisen, now NEU formally Rice
- Syntax
  
  \[
  M \rightarrow x \\
  \mid (\lambda.x \ M) \\
  \mid (M \ M) \\
  \mid n \\
  \mid (+ \ M \ M)
  \]

  How is it different from other languages?

Examples

- What does the following LC expressions return

  (A) 1
  
  (B) x
  
  (C) (\lambda.x \ x)
  
  (D) ( (\lambda.x \ x) \ 1)
  
  (E) ( (\lambda.f \ (f \ 1)) \ (\lambda.x \ x) )
Dr. Scheme
http://www.plt-scheme.org/

How to Design Programs
http://www.htdp.org/

;; How to design a program
(define (profit ticket-price)
  (+ (revenue ticket-price)
      (cost ticket-price)))

(define (revenue ticket-price)
  (* (attendees ticket-price) ticket-price))

(define (cost ticket-price)
  (+ 180
      (* .04 (attendees ticket-price)))))

(define (attendees ticket-price)
  (+ 120
      (* (/ 15 .10) (- 5.00 ticket-price))))

;; How not to design a program
(define (profit price)
  (- (* (+ 120
         (* (/ 15 .10)
            (- 5.00 price)))
      180
      (* .04
         (+ 120
            (* (/ 15 .10)
               (- 5.00 price)))))))

Figure 5: Two ways to express the profit program
Functional Programming

• Program to produce functions rather than data
  • what is an object?
• Communicate functions rather data
  • what is communicated on the Internet?
• Example
  • write a function `addk`
    • given k, produces a unary function that adds k to its input

Binding and Closure

• Variable binding
  • binding happens at the function application
  • use the closest binding under the static scope
• Closure
  • a function and its environment
    • the environment stores the value of all bound variables
• Implementation
  • can functions share a stack?
  • how to model memory?
    • when/how to free memory?
• Examples
  (A) `(λ.x x) 1`
  (B) `(λ.λ.f (f 1)) (λ.x x)`
Church Numerals

• Take f and x, number k is a function that apply f k times to x
  • zero is
  • one is
  • k is
• Operations
  • inc
  • add
  • multiply

Expanding the Language

• Global variables
• Implementing “define”
• Multiple parameters
Control Flow

- Branches
- Recursion

Semantics

How to define the complete meaning of a programming language? How long/short is the definition?
LC Representation in Scheme

(define-struct var (name))
(define-struct const (number))
(define-struct proc (param body))
(define-struct app (rator rand))
(define-struct add (left right))

1
(define one (make-const 1))

x
(define x (make-var 'x))
(\x. x x)
(define ifunc (make-proc 'x x))
( (\x. x x) 1)
(make-app ifunc one)

Environment

• An evaluation environment is a set of variable bindings

(define empty-env
 (lambda ()
   (lambda (var)
     (error 'lookup "variable \a not found" var))))

(define lookup
 (lambda (var env)
   (env var)))

(define extend
 (lambda (env varN val)
   (lambda (name)
     (if (eq? name varN)
       val
       (env name))))))
Closure

- A closure is a function and its evaluation environment

Semantics

- $MEval(M, ...)$
  - case $n$:
  - case $x$:
  - case $(\lambda x M)$:
  - case $(M_1, M_2)$: