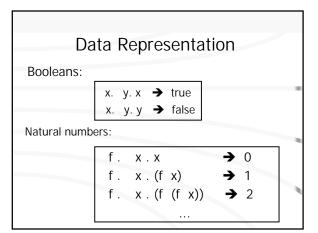
## **CSE 505**

Lecture #8

September 26, 2012



# let if = b. t. e.((b t) e)

We can justify the above reptn by showing:

- a. (((if true) T1) T2) ==>\* T1
- b. (((if false) T1) T2) ==> $^*$  T2

#### Example:

((( b. t. e.((b t) e) true) T1) T2)
==> (( t. e.((true t) e) T1) T2)
==>\* ((true T1) T2) = (( x. y.x T1) T2)
==> ( y.T1 T2)
==> T1

### Idea behind Church Numerals

Constructors: zero, succ(zero), succ(succ(zero)),

...

Alternatively: z, s(z), s(s(z)), ...

Lisp Syntax: z, (s z), (s (s z)), ...

Abstract Names: s. z.z,

s. z.(s z),

s. z.(s (s z)), ...

CSE 505 / Jayaraman

## Operations on numbers

Let succ = n. f. x. ((n f) (f x))

Let add = n1. n2. f. x.  $((n1 \ f) \ ((n2 \ f) \ x))$ 

Let mult = n1. n2. f. x. ((n1 (n2 f)) x)

Let mystery = n1. n2. (n2 n1)

(SUCC S. Z.Z)

#### **Data Structures**

Recall Lisp lists:

(cons 1 nil)

'(1 2 3) (cons 1 (cons 2 (cons 3 nil)))

The names of the constructors nil and cons are not important, so we "abstract them away" in lambda calculus, as shown on next slide.

Lecture 8: 9/26/ 2012

CCC FOF (I

### **Encoding Lists**

- c. n.n
- c. n. ((c tom) n)
- c. n . ((c tom) ((c dick) n))
- c. n . ((c tom) ((c dick) ((c harry) n)))
- •

Function to get first element:

I.((I x. y.x) a)

(I.((I x. y.x) a) c. n.((c tom) ((c dick) n)))

→\* tom

```
(I.((I x. y.x) a) c. n.((c tom) ((c dick) n)))
```

 $\Rightarrow$  (( c. n.((c tom) ((c dick) n))) x. y.x) a)

 $\Rightarrow$  ( n.(( x. y.x tom) (( x. y.x dick) n))) a)

 $\Rightarrow$  ( n.( y.tom (( x. y.x dick) n))) a)

 $\Rightarrow$  ( y.tom (( x. y.x dick) a)))

⇒tom

Lecture 8: 9/26/2012

CSE 505 / Jayaraman

### LAMBDA CALCULUS TOOL DEMO

Lecture 8: 9/26/ 2012

CSE 505 / Jayaraman

## Computability

- The language of lambda expressions is powerful enough to encode all computable functions!
- Notice that there is no recursive function definition – but this can be simulated, as will be next shown.

### **Recursive Definition**

Consider recursive definition:

f(n) = if isO(n) then 1 else n \* f(n-1)

Lisp syntax:

(defun f (n) (if (is0 n) 1 (\* n (f (- n 1)))))

Lambda calculus (not quite):

letrec f = n.(((if (is0 n)) 1) ((mult n) (f (pred n))))

Lecture 8: 9/26/ 2012

12

## Representing Recursion

$$\begin{array}{ll} \text{let } t = & \text{f.} & \text{n.}(((\text{if (is0 n)}) \ \ 1) \\ & & ((\text{mult n}) \ (\text{f (pred n)}))) \end{array}$$

Fixed-Point Operator, Y:

let 
$$Y = f. (x.(f(x x)) (x.(f(x x)))$$

Note: Fixed point of F is an x such that F(x) = x

Thus the non-recursive equivalent of fact: (Y t)

$$Y = f. (x.(f(x x)) x.(f(x x)))$$

Y is fixed-point operator, because (for any t):

$$(Y \ t) <==>^* (t \ (Y \ t))$$

Derivation:

$$(Y t) = (f. (x.(f (x x)) x.(f (x x))) t)$$

$$==> (x.(t (x x)) x.(t (x x)))$$

$$==> (t (x.(t (x x)) x.(t (x x)))))$$

$$<==> (t (Y t))$$

Lecture 8: 9/26/ 2012

COP FOR I

## Recursion and Fixed-points

fact = n.(((if (is0 n)) 1) ((mult n) (fact (pred n)))

$$t = f. n.(((if (is0 n)) 1) ((mult n) (f (pred n))))$$

Why does the fixed-point of t capture f?

Fixed point g has the property: g = (t g)

$$g = n.(((if (is0 n)) 1) ((mult n) (g (pred n)))$$

Lecture 8: 9/26/2012

CSE 505 / Javaraman

#### Least Fixed Point

Consider: letrec f(n) = if n=0 then 0 else f(n);

Fixed-point f1(n) = 
$$\begin{cases} 0, & \text{if } n=0 \\ 1, & \text{if } n=0 \end{cases}$$
Fixed-point f2(n) = 
$$\begin{cases} 0, & \text{if } n=0 \\ 2, & \text{if } n=0 \end{cases}$$

Least fixed-point 
$$g(n) = \begin{cases} 0, & \text{if } n=0 \\ ?, & \text{if } n=0 \end{cases}$$

Lecture 8: 9/26/ 2012

CSE 505 / Jayaraman

## Typed Lambda Calculi

Thus far, we have studied the untyped lambda calculus, i.e., no types associated with vars.

There are two well-known calculi:

- the simply-type lambda calculus
- the second-order (polymorphic) lambda calculus

Interesting, adding types causes all lambda expressions to terminate! Cannot have (x x).

Lecture 8: 9/26/2012

CSE 505 / Jayaraman