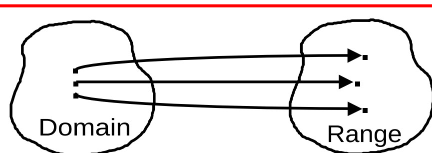
λ-calculus: A Basis for Functional Languages

Arvind
Laboratory for Computer Science
M.I.T.

September 11, 2002

http://www.csg.lcs.mit.edu/6.827

Functions



f : D →?*R*

f may be viewed as

- a set of ordered pairs < *d* , *r* > where *d* ε *D* and *r* ε *R*
- a *method of computing* value *r* corresponding to argument *d*

some important notations

- λ-calculus (Church)
- Turing machines (Turing)
- Partial recursive functions

The λ-calculus: a simple type-free language

- to express *all computable functions*
- to directly *express higher-order functions*
- to study *evaluation orders*, *termination*, *uniqueness of answers*…
- to study various *typing systems*
- to serve as *a kernel language for functional languages*
  - However, λ-calculus extended with constants and let-blocks is more suitable

λ-notation

- a way of writing and applying functions without having to give them names
- a syntax for making a function expression from any other expression
- the syntax distinguishes between the *integer "2"* and the *function "always_two"* which when applied to any integer returns 2

```
always_two x = 2;
```

L3-5
Arvind

Pure λ-calculus: Syntax

E = x | λx.E | E E

variable   abstraction   application

1. *application* $E_1\ E_2$

function   argument

- application is left associative

  $E_1\ E_2\ E_3\ E_4 \equiv (((\ E_1\ E_2\ )\ E_3\ )\ E_4\ )$

2. *abstraction*   λx.E

bound variable or formal parameter   body

- the scope of the dot in an abstraction extends as far to the right as possible

  λx.x y ≡ λx.(x y) ≡?(λx.(x y)) ≡ (λx.x y) ≠?(?λx.x) y

L3-6
Arvind

Free and Bound Variables

- λ-calculus follows *lexical scoping* rules
- *Free variables* of an expression

  FV(x) = {x}
  $FV(E_1\ E_2) = FV(E_1)\ U\ FV(E_2)$
  FV(λx.E) = FV(E) - {x}

- A variable occurrence which is not free in an expression is said to be a *bound variable* of the expression
- *combinator:* a λ-expression without free variables,

  aka *closed λ-expression*

ß-substitution

$(\lambda x.E)\ E_a \rightarrow E[E_a/x]$
replace all free occurrences of x in E with E_a

E[A/x] is defined as follows by case on E:

variable

$y[E_a/x] = E_a$	if $x \equiv y$
$y[E_a/x] = y$	otherwise

application

$(E_1\ E_2)[E_a/x] = (E_1[E_a/x]\quad E_2[E_a/x])$

abstraction

$(\lambda y.E_1)[E_a/x]$	$= \lambda y.E_1$	if $x \equiv y$
$(\lambda y.E_1)[E_a/x]$	$= \lambda z.((E_1[z/y])[E_a/x])$	otherwise
	where $z \notin FV(E_1) \cup FV(E_a) \cup FV(x)$	

ß-substitution: an example

$(\lambda p.p\ (p\ q))\ [(a\ p\ b)\ /\ q]$

L3-9
Arvind

λ-Calculus as a Reduction System

Syntax

$E = x \mid \lambda x.E \mid E\ E$

Reduction Rule

α-*rule:* $\lambda x.E \rightarrow \lambda y.E\ [y/x]$ if $y \notin FV(E)$

β-*rule:* $(\lambda x.E)\ E_a \rightarrow E\ [E_a/x]$

η-*rule:* $(\lambda x.E\ x) \rightarrow E$ if $x \notin FV(E)$

Redex

$(\lambda x.E)\ E_a$

Normal Form

An expression without redexes

L3-10
Arvind

α and η Rules

α-rule says that the bound variables can be renamed systematically:

$(\lambda x.x\ (\lambda x.a\ \ x))\ b \equiv (\lambda y.y\ \ (\lambda x.a\ \ x))\ b$

η-rule can turn any expression, including a constant, into a function:

$\lambda x.a\ x \rightarrow_{\eta} a$

η-rule does not work in the presence of types

A Sample Reduction

C ≡ λx.λy.λf.f x y
H ≡ λf.f (λx.λy. x)
T ≡ λf.f (λx.λy. y)

What is H (C a b) ?

Integers: Church's Representation

0 ≡ λx.λy. y
1 ≡ λx.λy. x y
2 ≡ λx.λy. x (x y)
...
n ≡ λx.λy. x (x...(x y)...)

succ ?

If *n* is an integer, then *(n a b)* gives *n* nested *a*'s followed by *b*

⇒ the successor of *n* should be *a (n a b)*

succ ≡ λn.λa.λb.a (n a b)
plus ≡ λm.λn.m succ n
mul ≡ ?

L3-13
Arvind

Booleans and Conditionals

True ≡ $\lambda x.\lambda y.x$
False ≡ $\lambda x.\lambda y.y$

zero? ≡ $\lambda n.$ n ($\lambda y.$False) True
zero? 0 $\twoheadrightarrow$?

zero? 1 $\twoheadrightarrow$?

cond ≡ $\lambda b.\lambda x.\lambda y.$ b x y
cond True E_1 E_2 $\twoheadrightarrow$?
cond False E_1 E_2 $\twoheadrightarrow$?

L3-14
Arvind

Recursion ?

```
fact n = if (n == 0) then 1
                     else n * fact (n-1)
```

- Assuming suitable combinators, fact can be rewritten as:

  fact = $\lambda n.$ cond (zero? n) 1 (mul n (fact (sub n 1)))

- *How do we get rid of the* fact *on the RHS?*

Choosing Redexes

1. ((λx.M) A) ((λx.N) B)
   ------ ρ_1------   ------ ρ_2------

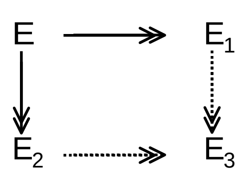
2. ((λx.M) ((λy.N)B))
   ------ ρ_2------
   ------------- ρ_1-----------

Does ρ_1 followed by ρ_2?produce the same expression as ρ_2 followed by ρ_1?

Notice in the second example ρ_1 can *destroy* or *duplicate* ρ_2 .

Church-Rosser Property

A reduction system is said to have the *Church-Rosser property*, if $E \twoheadrightarrow E_1$ and $E \twoheadrightarrow E_2$ then there exits a E_3 such that $E_1 \twoheadrightarrow E_3$ and $E_2 \twoheadrightarrow E_3$.



also known as *CR* or *Confluence*

Theorem: The λ-calculus is CR.
(Martin-Lof & Tate)

Interpreters

An *interpreter* for the λ-calculus is a program to reduce λ-expressions to “answers”.

It requires:

- the definition of an *answer*
- a *reduction strategy*
    - a method to choose redexes in an expression
- a criterion for *terminating* the reduction process

Definitions of “Answers”

- *Normal form (NF):* an expression without redexes
- *Head normal form (HNF):*
  - x is HNF
  - (λx.E) is in HNF if E is in HNF
  - (x E_1 ... E_n) is in HNF

    Semantically most interesting- represents the information content of an expression
- *Weak head normal form (WHNF):*

  An expression in which the left most application is not a redex.
  - x is in WHNF
  - (λx.E) is in WHNF
  - (x E_1 ... E_n) is in WHNF

    Practically most interesting ⇒ “Printable Answers”

L3-19
Arvind

Reduction Strategies

There are many methods of choosing redexes in an expression

((λx.M) ((λy.N)B))
------ ρ_2------
-------------- ρ_1 ------------

- *applicative order:* left-most innermost redex
  - would reduce ρ_2 before ρ_1
- *normal order:* left-most (outermost) redex
  - would reduce ρ_1 before ρ_2

L3-20
Arvind

Facts

1. Every λ-expression does not have an answer *i.e.,* a NF or HNF or WHNF

   (λx. x x) (λx. x x) = Ω

   Ω →

2. CR implies that if NF exists it is *unique*

3. Even if an expression has an answer, not all *reduction strategies* may produce it

   (λx.λy. y) Ω

   leftmost redex: (λx.λy. y) Ω → λy. y
   innermost redex: (λx.λy. y) Ω →

Normalizing Strategy

A *reduction strategy* is said to be *normalizing* if it terminates and produces an answer of an expression whenever the expression has an answer.

aka *the standard reduction*

Theorem: Normal order (left-most) reduction strategy is normalizing for the λ-calculus.

A Call-by-name Interpreter

Answers: WHNF
Strategy: leftmost redex

cn(E): Definition by cases on E

E = x | λx. E | E E

$cn(x) = x$
$cn(\lambda x.E) = \lambda x.E$
$cn(E_1\ E_2) =$ *let* $f = cn(E_1)$
in
case f *of*
$\lambda x.\ E_3 = cn(E_3[E_2/x])$
$\_ = (f\ E_2)$

L3-23
Arvind

A Call-by-value Interpreter

Answers: WHNF
Strategy: leftmost-innermost redex but not inside a λ-abstraction

cv(E): Definition by cases on E

E = x | λx. E | E E

```
cv(x)       = x
cv(λx.E)    = λx.E
cv( E1 E2 ) = let f = cv(E1)
                  a = cv(E2)
              in
                  case f of
                     λx. E3 = cv(E3[a/x])
                     _      = (f a)
```


L3-24
Arvind

More Facts

For computing WHNF

the call-by-name interpreter is normalizing but the call-by-value interpreter is not

e.g.
(λx.y) ((λx.x x) (λx.x x))