Principles of Programming Languages

CS 314

Recitation 9

Topics Today

Lambda Calculus

β-Reduction

α-Reduction

Programming in Lambda Calculus

Lambda Calculus

β-Reduction

α-Reduction

Programming in Lambda Calculus

Lambda Calculus

- A unified language to manipulate and reason about functions.
- Consider the function $f(x) = x + 4$.
- This can be expressed as $\lambda x.(+\ x\ 4)$
- By applying a value to the function, we can evaluate the function.
- $((\lambda x.(+\ x\ 4))\ 2) = 6$ is equivalent to saying $f(2) = 2 + 4 = 6$.

Expressions in Lambda Calculus consist of λ-terms.

- Variables such as “x” or values such as “2”.
- Function abstraction (λx. M), where M is some λ-term.
- Function application (M N), where M and N are λ-terms.

An example of function application was: ((λx.(+ x 4)) 2)

- M = (λx.(+ x 4))
- N = 2

Function abstraction

- We can express a function of many arguments: $f(x, y) = x + y + 1$:

$$(\lambda x.\lambda y.(+\ (+\ x\ y)\ 1))$$

- We can also write the above expression as

$$(\lambda xy.(+\ (+\ x\ y)\ 1))$$

- One lambda for each input variable "$\lambda x.\lambda y.$", or one letter for each variable "$\lambda xy.$".

Free and bound variables

If we have an expression as follows:

$$(\lambda x.\ M)$$

Then we say that x is bound in expression M. All other variable occurrences in M for this particular function are free.

Free and bound variables

What variables are bound in this expression?

$$(\lambda xy.((x\ y)\ (x\ w)))$$

Free and bound variables

What variables are bound in this expression?

$$(\lambda xy.((x\ y)\ (x\ w)))$$

x and y are bound in this expression.

What variables are free in this expression?

Free and bound variables

What variables are bound in this expression?

$$(\lambda xy.((x\ y)\ (x\ w)))$$

x and y are bound in this expression.

What variables are free in this expression?

w is free in this expression.

Lambda Calculus

β-Reduction

α-Reduction

Programming in Lambda Calculus

Beta-Reduction

β-Reduction is the technique of applying functions to their arguments.

$$((\lambda x.M)\ v) = [v\ /\ x]M$$

The notation "[v / x]M" means replacing all free occurrences of x in M with v.

Examples:

- ((λx.(+ x 1)) 2) = [2 / x](+ x 1) = (+ 2 1) = 3
- ((λx.(+ x x)) 2) = (+ 2 2) = 4
- ((λx.3) 2) = 3 No substitution happens here.

Beta-Reduction

More examples:

- ((λx.λy.(+ x y)) 2) = [2 / x](λy.(+ x y)) = (λy.(+ 2 y)) = 2 + y
- Apply the outer-most λ first.

- ((λx.(x y)) (λz.z)) = ((λz.z) y) = y.
- Functions can be “values” that can be applied.

- What about ((λx.λy.xy) (λz.(z z)) x)?

Beta-Reduction

More examples:

- $((\lambda x.\lambda y.(+\ x\ y))\ 2) = [2\ /\ x](\lambda y.(+\ x\ y)) = (\lambda y.(+\ 2\ y)) = 2 + y$
- Apply the outer-most λ first.

- $((\lambda x.(x\ y))\ (\lambda z.z)) = ((\lambda z.z)\ y) = y.$
- Functions can be “values” that can be applied.

- $((\lambda x.\lambda y.xy)\ (\lambda z.(z\ z))\ x) = ((\lambda y.((\lambda z.(z\ z))\ y)\ x)$ Substitute (λz.(z z)) for x
  $= ((\lambda z.(z\ z))\ x)$ Substitute x for y
  $= (x\ x)$ Substitute x for z

Lambda Calculus

β-Reduction

α-Reduction

Programming in Lambda Calculus

Alpha-Reduction

Suppose we have the following example:

$$((\lambda x.\lambda y.(x\ y))\ y\ w)$$

If I used β-Reduction here, I would've gotten

$$((\lambda y.(y\ y))\ w)$$

Even though y is not equal to y. How do we fix this problem?

We can use α-Reduction, which first renames y into another variable, say z.

$$((\lambda x.\lambda z.(x\ z))\ y\ w)$$

Now we can continue with β-Reduction.

$$((\lambda x.\lambda z.(x\ z))\ y\ w) = ((\lambda z.(y\ z))\ w) = (y\ w)$$

Alpha-Reduction

Another example: Use α-Reduction and β-Reduction to reduce this expression:

$$((\lambda x.\lambda y.(x\ y))\ (\lambda y.\ y)\ w)$$

Alpha-Reduction

Another example: Use α-Reduction and β-Reduction to reduce this expression:

$$((\lambda x.\lambda y.(x\ y))\ (\lambda y.\ y)\ w)$$

First, use α-Reduction to rename y

$$((\lambda x.\lambda z.(x\ z))\ (\lambda y.\ y)\ w)$$

Now use β-Reduction

$$((\lambda x.\lambda z.(x\ z))\ (\lambda y.\ y)\ w) = ((\lambda z.((\lambda y.\ y)\ z))\ w)$$

$$((\lambda z.((\lambda y.\ y)\ z))\ w) = ((\lambda y.\ y)\ w)$$

$$((\lambda y.\ y)\ w) = w$$

Lambda Calculus

β-Reduction

α-Reduction

Programming in Lambda Calculus

Programming in Lambda Calculus

We can also program using Lambda Calculus.

First, some definitions:

True = (λxy.x) (select-first)
False = (λxy.y) (select-second)

and = (λxy.((x y) False))
not = (λx.((x False) True))

Can we show that (not False) = True?

We want to show that (not False) = True:

Recall that not = $(\lambda x.((x \text{ False}) \text{ True}))$ and False = $(\lambda xy.y)$

$$(\text{not False}) = ((\lambda x.((x \text{ False}) \text{ True})) \text{ False})$$

By β-Reduction we have:

$$(\text{not False}) = ((\text{False False}) \text{ True}) = ((\lambda xy.y) \text{ False}) \text{ True})$$

By β-Reduction, again, we have:

$$((\lambda xy.y) \text{ False}) \text{ True}) = ((\lambda y.y) \text{ True}) = \text{True}$$

Thus, (not False) = True

Programming in Lambda Calculus

Given

True = (λxy.x) (select-first)

False = (λxy.y) (select-second)

and = (λxy.((x y) False))

not = (λx.((x False) True))

Can we show that (and True False) = False?

We want to show that (and True False) = False:

Recall that and = (λxy.((x y) False)) and True = (λxy.x)

(and True False) = ((λxy.((x y) False)) True False)

By β-Reduction we have:

(and True False) = ((True False) False) = ((λxy.x) False) False)

By β-Reduction, again, we have:

((λxy.x) False) False) = ((λy.False) False) = False

Thus, (and True False) = False.

Consider: What would be the Lambda Calculus representation of “or”?

Church numerals

$0 = (\lambda f.\lambda x.x)$
$1 = (\lambda f.\lambda x.(f\ x))$
$2 = (\lambda f.\lambda x.(f\ (f\ x)))$
…
$n = (\lambda f.\lambda x.(f\ (f\dots\ (f\ (f\ x)))$ n copies of “f”

So, (n f x) = (f (f (f… f (f x)))), with n copies of “f”, where n is the church numeral.

Church numerals

The successor function ($s(x) = x + 1$) is defined as follows:

$$s(n) = (\lambda n.\lambda fx.(f\ (n\ f\ x)))$$

Note that the idea is to just apply one more "f"

Example: $s(1) = 2$: Recall that $1 = (\lambda f.\lambda x.(f\ x))$

$$((\lambda n.\lambda fx.(f\ (n\ f\ x)))\ (\lambda f.\lambda x.(f\ x))) = (\lambda fx.(f\ ((\lambda f.\lambda x.(f\ x))\ f\ x)))$$
$$= (\lambda fx.(f\ (f\ x))) = 2$$

Church numerals

The addition function (add(m, n) = m + n) is defined as follows:

$$\text{add}(m, n) = (\lambda mn.\lambda fx.(m\ f\ (n\ f\ x)))$$

Example: add(1, 2) = 3: Recall that 1 = (λfx.(f x)) and 2 = (λfx.(f (f x)))

add(1, 2) = ((λmn.λfx.(m f (n f x))) (λfx.(f x)) (λfx.(f (f x))))

= (λfx.((λfx.(f x)) f ((λfx.(f (f x))) f x)))

= (λfx.((λx.(f x)) ((λfx.(f (f x))) f x)))

= (λfx.((λx.(f x)) (f (f x))))

= (λfx.(f (f (f x)))) = 3.

Church numerals

The multiplication function (mult(m, n) = m * n) is defined as follows:

$$\text{mult}(m, n) = (\lambda mn.\lambda fx.((m\ (n\ f))\ x)$$

Example: mult(1, 2) = 2: Recall that $1 = (\lambda fx.(f\ x))$ and $2 = (\lambda fx.(f\ (f\ x)))$

$$\text{mult}(1, 2) = ((\lambda mn.\lambda fx.((m\ (n\ f))\ x)\ (\lambda fx.(f\ x))\ (\lambda fx.(f\ (f\ x))))$$

$$= (\lambda fx.(((\lambda fx.(f\ x))\ ((\lambda fx.(f\ (f\ x)))\ f))\ x)$$

$$= (\lambda fx.(((\lambda fx.(f\ x))\ (\lambda x.(f\ (f\ x)))))\ x)$$

$$= (\lambda fx.(((\lambda x.((\lambda x.(f\ (f\ x)))\ x))))\ x)$$

$$= (\lambda fx.((\lambda x.(f\ (f\ x)))\ x) = (\lambda fx.(f\ (f\ x))) = 2.$$