Lambda Calculus I

4190.310
Programming Languages
Spring 2014

Lecture 02

Reading assignments: Chapter 10

Backus-Naur Form (BNF)

- A notation for context-free grammar
- Used to describe the syntax of programming languages
- Start with a symbol and then replace this symbol according to given rules
  - Production rules
- *symbol* ::= *expression*
  - Terminals and non-terminals
    - *symbol*: non-terminal (the symbol on the left of the rule)
    - x: terminal
    - 1: terminal
  - ::=: symbol is replaced with *expression*
  - *expression*: one or more sequences of symbols
  - |: choice

Functions

- A function is a mapping that associates to each x from some set X of values a unique y from a set of Y values
  - y = f(x)
  - f: X → Y
    - X: domain of f
    - Y: range of f
- When f is not defined for all x in X, it is called a partial function

Function Definitions and Function Applications

- Function definitions
  - Describes how a value is to be computed using formal parameters
  - sqrt(x) = x * x
- Function applications
  - A call to a defined function using actual parameters
  - sqrt(2)

Functions as First-class Data Values

- A function is viewed as a value itself
  - The value can be computed by other functions
  - The function itself can be a parameter to functions
- Higher-order functions
  - Have parameters that are themselves functions or that produce a result that is a function, or both

λ-Calculus

- Developed by Alonzo Church
- A notation for arbitrary functions
  - Computable functions
  - Church-Turing thesis
    - The functions that can be calculated using a mechanical calculation device given unlimited amounts of time and storage
- Lisp (McCarthy 1960) closely resembles λ-calculus
- In lambda calculus, functions can take functions as arguments and return functions as results
  - Every value is a function in pure lambda calculus

Syntax (Pure λ-calculus)

- BNF grammar for λ-terms
- Inductive definition
  - A variable is a λ-term
  - If t is a λ-term, then λx.t is a λ-term (λ abstraction)
  - If t and s are λ-terms, ts is a λ-term (function application)
    - Left associative

$$
\begin{array}{lll}
M ::= x & & \text{(variable)} \\
 & \mid MM & \text{(function application)} \\
 & \mid \lambda x.M & \text{(}\lambda\text{ abstraction)}
\end{array}
$$

Unambiguous Grammar

<term> ::= <atom> | <app> | <fun>
<atom> ::= <h-atom> | (<app>)
<h-atom> :: = x | (<fun>)
<app> ::= <h-atom> <atom> | <app> <atom>
<fun> ::= λx.<term>

λ-term Examples

- Examples below are not in pure λ-calculus
  - Arithmetic expressions and constants
- λx.x
  - Identity function
- (λx.x) 6
  - Function application
- λy. x y
- 5
- λf.λg.λx. f(gx)

Functions in Pascal and LISP

- λ x . 3 * x
- function f(x : int) : int begin f := 3 * x end;
- (lambda (x) (* 3 x))

Free and Bound Variables

- λ: binding operator
  - Binds a variable within a specific scope
- Free variable in an expression
  - The variable is not declared in the expression
- Bound variable in an expression
  - The variable is not free in the expression
- Examples
  - λx.xy
  - (λx.x)(λy.yx)

Substitution

- $[z/x]\ M$
  - The result of substituting y for free occurrences of x in M
  - z cannot already appear in M

- Inductive definition
  - $[N/x]x = N$ *for any variable* x
  - $[N/x]y = y$ *when* y *is any variable different from* x
  - $[N/x](M\ P) = ([N/x]M)([N/x]P)$
  - $[N/x](\lambda x.M) = \lambda x.M$
  - $[N/x](\lambda y.M) = \lambda y.([N/x]M)$ *when* $x \neq y$ *and* y *is not free in* N
  - $[N/x](\lambda y.M) = \lambda z.([N/x]([z/y]M))$ when $x \neq y$ *and* y *is free in* N*, where* $x \neq z$ *and* z *is not free in* N *and* M

Some Rules

- Function application associates to the left
  - MNP = (MN)P
- The scope of a variable specified by λ extends as far to the right as possible
  - λx.MNP = λx.(MNP) ≠(λx.MN)P
- A sequence of variables specified by a λ abbreviates a series of λ abstractions
  - λxyz.M = λx.(λy.(λz.M))

α-equivalence

- A choice of a bound variable in a λ abstraction is not important
- λx.M = λy.[y/x]M *when y is not free in M*

β-equivalence

- To calculate the value of a function application by substitution
  - Replacement of a formal parameter by an actual parameter
  - Textual replacement
- (λx.M)N = [N/x]M
  - Rename any bound variable in M that might be the same as free variables in N
    - Do not mix up free occurrences of a variable with its bound occurrences
  - A term of the form (λx.t1) t2 is called a β-redex
- Left associative
  - MNP = (MN)P

β-equivalence (contd.)

- Examples
  - (λx. (3+x)) 5 = 3+x
  - (λy.y 5) (λx.3+x) = (λx.3+x) 5 = 3+5
  - (λf.λx.f x)(λy.y+x) = λx.((λy.y+x) x) = λx.x+x
    - A free variable x in λy.y+x becomes bound in λx.((λy.y+x) x)
      - Incorrect
  - (λf.λx.f x)(λy.y+x) = (λf.λz.f z)(λy.y+x) = λz.((λy.y+x) z) = λz.z+x

β-equivalence (contd.)

- More examples
  - $\underline{(\lambda x.\lambda y.y\ x)(\lambda z.u)} = \lambda y.y\ (\lambda z.u)$
  - $\underline{(\lambda x.x\ x)(\lambda z.u)} = (\lambda z.u)\ (\lambda z.u)$
  - $(\lambda y.y\ v)\underline{((\lambda x.x)(\lambda z.(\lambda u.u)\ z))} = (\lambda y.y\ v)(\lambda z.(\lambda u.u)\ z)$
  - $(\lambda y.y\ v)((\lambda x.x)(\lambda z.\underline{(\lambda u.u)\ z})) = (\lambda y.y\ v)((\lambda x.x)(\lambda z.z))$
  - $\underline{(\lambda y.y\ v)((\lambda x.x)(\lambda z.(\lambda u.u)\ z))} = ((\lambda x.x)(\lambda z.(\lambda u.u)\ z))\ v$

Normal Form

- If a sequence of reductions has come to an end where no further reductions are possible, we say that the term has been reduced to normal form
  - A term containing no β-redexes

- Not every term has a normal form
  - $(\lambda x.xx)\ (\lambda x.xx)$

- Normal order
  - Reduce the leftmost-outermost redex

Church-Rosser Theorem

- The result of a computation is independent from the the order of reduction
- [Theorem] If a term M can be reduced to terms N and P in 0 or more steps, then there exists a term Q to which both N and P can be reduced in 0 or more steps
  - Diamond property
- [Corollary]
  - Every λ-term has at most one normal form

$$\begin{tikzcd}
& M \arrow[dl, "*"'] \arrow[dr, "*"] & \\
N \arrow[dr, "*"'] & & P \arrow[dl, "*"] \\
& Q &
\end{tikzcd}$$