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)
      grammer symbol?
    - 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
      왼쪽에서부터 계산해 나간다.

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

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
  parameter : x
  - Identity function
- (λx.x) 6
  - Function application
- λy. x y
- 5
- λf.λg.λx. f(gx)

M :: = x
| MM
| lambda x.M

Functions in Pascal and LISP

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

  parameter : type result type
- (lambda (x) (* 3 x)) lisp

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 x는 binding, y는 free
  - (λx.x)(λy.yx) 앞에 x는 binding, 뒤에 x는 free

Substitution

/ : replace, x를 z로 replace하라!, free인 경우만 가능, binding은 안댐ㅋ

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

- Inductive definition
  - $[N/x]_x = N$
  - [N/x]y = y, *where y is any variable different from x*
  - [N/x](M P) = ([N/x]M)([N/x]P)
  - [N/x](λx.M) = λx.M
  - [N/x](λy.M) = λy.([N/x]M), where y is not free in N

Some Rules

- Function application associates to the left left associative
  - 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 줄여서 쓰다
  - λxyz.M = λx.(λy.(λz.M))

α-equivalence

- A choice of a bound variable in a λ abstraction is not important

- $\lambda x.M = \lambda.y.[y/x]M$

β-equivalence

- To calculate the value of a function application by substitution
  - Replacement of a formal parameter by an actual parameter
  - Textual replacement

- $(\lambda x.M)N = [N/x]M$
  - 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 $(\lambda x.t1)\ t2$ is called a β-redex

reduction과정에서 선택받은 녀석

- Left associative
  - MNP = (MN)P

β-equivalence (contd.)

- Examples
  - $(\lambda x.\ (3 + x))\ 5 = (3 + x)$ 3+5(lambda calculus reduction) = 8(arithmatic)
  - $(\lambda y.y\ 5)\ (\lambda x.3+x) = (\lambda x.3+x)\ 5 = 3 + 5$
  - $(\lambda f.\lambda x.f\ x)(\lambda y.y + x) = \lambda x.((\lambda y.y + x)\ x) = \lambda x.x + x$

    free인 x가 bound가 되었기 때문에 이 과정 자체가 안됨.

    free bound
    - A free variable x in $\lambda y.y + x$ becomes a bound variable in $\lambda x.x + x$
  - $(\lambda f.\lambda x.f\ x)(\lambda y.y + x) = (\lambda f.\lambda z.f\ z)(\lambda y.y + x) = \lambda z.((\lambda y.y + x)\ z) = \lambda z.z + x$

    (lambda y.(y+x))

β-equivalence (contd.)

- More examples
  - $(\lambda x.\lambda y.y\ x)(\lambda z.u) = \lambda y.y\ (\lambda z.u)$ not equal : (labmda y.y) (lamda z.u)
  - $(\lambda x.x\ x)(\lambda z.u) = (\lambda z.u)\ (\lambda z.u)$
  - $(\lambda y.y\ v)((\lambda x.x)(\lambda z.(\lambda u.u)\ z)) = (\lambda y.y\ v)(\lambda z.(\lambda u.u)\ z)$
  - $(\lambda y.y\ v)((\lambda x.x)(\lambda z.(\lambda u.u)\ z)) = (\lambda y.y\ v)((\lambda x.x)(\lambda z.z))$
  - $(\lambda y.y\ v)((\lambda x.x)(\lambda z.(\lambda u.u)\ z)) = ((\lambda x.x)(\lambda z.(\lambda u.u)\ z))\ v$

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
  - (λx.xx) (λx.xx)
- Normal order
  - Reduce the leftmost-outermost redex

Church-Rosser Theorem

- The result of a computation is independent from 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] 따름 정리 : 기존 정리를 이용하여 만든 정리
  - Every λ-term has at most one normal form