Lambda Calculus

John Mitchell

Reading: Chapter 4

Lambda Calculus

- Formal system with three parts
  - Notation for defining functions
  - Proof system for proving equations
  - Calculation rules called *reduction*

There is more detail in the book than we will cover in class

History

- Original intention
  - Foundations of mathematics (1930 - Church)
- More successful for computable functions
  - Substitution --> symbolic computation
  - Church/Turing thesis
- Influenced design of Lisp, ML, other languages
  - See Boost Lambda Library for C++ function objects
- Important part of CS history and foundations

Why study this now?

- Basic syntactic notions
  - Free and bound variables
  - Functions
  - Declarations
- Calculation rule
  - Symbolic evaluation useful for discussing programs
  - Used in optimization (in-lining), macro expansion
    - Correct macro processing requires variable renaming
  - Illustrates some ideas about scope of binding
    - Lisp originally departed from standard lambda calculus, returned to the fold through Scheme, Common Lisp

Expressions and Functions

- Expressions

  $x + y$ $\qquad$ $x + 2*y + z$

- Functions

  $\lambda x. (x+y)$ $\qquad$ $\lambda z. (x + 2*y + z)$

- Application

  $(\lambda x. (x+y))\ 3 \quad = \quad 3 + y$

  $(\lambda z. (x + 2*y + z))\ 5 \quad = \quad x + 2*y + 5$

Parsing: $\lambda x.\ f\ (f\ x) = \lambda x.(\ f\ (f\ (x))\ )$

Higher-Order Functions

- Given function f, return function $f \circ f$

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

- How does this work?

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

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

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

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

Same result if step 2 is altered.

Same procedure, Lisp syntax

- Given function f, return function $f \circ f$

  (lambda (f) (lambda (x) (f (f x))))

- How does this work?

  ((lambda (f) (lambda (x) (f (f x)))) (lambda (y) (+ y 1))

  = (lambda (x) ((lambda (y) (+ y 1))
  ((lambda (y) (+ y 1)) x))))

  = (lambda (x) ((lambda (y) (+ y 1)) (+ x 1))))

  = (lambda (x) (+ (+ x 1) 1))

JavaScript next slide

Same procedure, JavaScript syntax

- Given function f, return function f ∘ f

```
function (f) { return function (x) { return f(f(x)); } ; }
```

- How does this work?

```
(function (f) { return function (x) { return f(f(x)); } ; )
    (function (y) { return y +1; })

function (x) { return (function (y) { return y +1; })
                               ((function (y) { return y + 1; })
(x)); }

function (x) { return (function (y) { return y +1; }) (x +
1); }

function (x) { return ((x + 1) + 1); }
```

Declarations as “Syntactic Sugar”

```
function f(x) {
    return x+2;
}
f(5);
```

$$\underbrace{(\lambda f.\ \ f(5))}_{\text{block body}}\ \ \underbrace{(\lambda x.\ x+2)}_{\text{declared function}}$$

$$\text{let } x = e_1 \text{ in } e_2 \quad = \quad (\lambda x.\ \ e_2)\ \ e_1$$

Free and Bound Variables

- Bound variable is "placeholder"
  - Variable x is bound in $\lambda x.\ (x+y)$
  - Function $\lambda x.\ (x+y)$ is same function as $\lambda z.\ (z+y)$
- Compare

  $\int x+y\ dx \;=\; \int z+y\ dz \qquad \forall x\ \ P(x) = \forall z\ \ P(z)$
- Name of free (=unbound) variable does matter
  - Variable y is free in $\lambda x.\ (x+y)$
  - Function $\lambda x.\ (x+y)$ is *not* same as $\lambda x.\ (x+z)$
- Occurrences
  - y is free and bound in $\lambda x.\ ((\lambda y.\ y+2)\ x) + y$

Reduction

- Basic computation rule is β-reduction

$$(\lambda x.\ e_1)\ e_2 \quad \to \quad [e_2/x]e_1$$

where substitution involves renaming as needed

(next slide)

- Reduction:
  - Apply basic computation rule to any subexpression
  - Repeat
- Confluence:
  - Final result (if there is one) is uniquely determined

Rename Bound Variables

- Function application

$$\underbrace{(\lambda f.\ \lambda x.\ f\ (f\ x))}_{\text{apply twice}}\ \ \underbrace{(\lambda y.\ y+x)}_{\text{add x to argument}}$$

- Substitute "blindly"

$$\lambda x.\ [(\lambda y.\ y+x)\ ((\lambda y.\ y+x)\ x)] \ =\ \lambda x.\ x+x+x$$

- Rename bound variables

$$(\lambda f.\ \lambda z.\ f\ (f\ z))\ \ (\lambda y.\ y+x)$$

$$=\ \lambda z.\ [(\lambda y.\ y+x)\ ((\lambda y.\ y+x)\ z))]\ =\ \lambda z.\ z+x+x$$

Easy rule: always rename variables to be distinct

Main Points about Lambda Calculus

- λ captures "essence" of variable binding
  - Function parameters
  - Declarations
  - Bound variables can be renamed
- Succinct function expressions
- Simple symbolic evaluator via substitution
- Can be extended with
  - Types
  - Various functions
  - Stores and side-effects

  (But we didn't cover these)