CSC 520, Spring 2020

The Lambda Calculus

Ravi Sethi

THE UNIVERSITY OF ARIZONA

Version: 1

February 17, 2020

Introduction

The Pure Untyped Lambda Calculus

The syntax has just three rules

- Syntax of Terms

$$
\begin{array}{lll}
M \rightarrow & x & \textit{variables} \\
\quad\mid & (\; M_1\, M_2 \;) & \textit{function application} \\
\quad\mid & (\; \lambda\;\; x\, .\, M \;) & \textit{abstraction}
\end{array}
$$

- Examples of terms
  - $(\lambda x.\ x)$
  - $(\lambda y.\ x)$
  - $(\lambda x.\ (\lambda y.\ x))$
  - $(\lambda x.\ (\mathrm{x}\ x))$

- What do you think these terms represent?

Lambda Calculus

Significance for programming languages

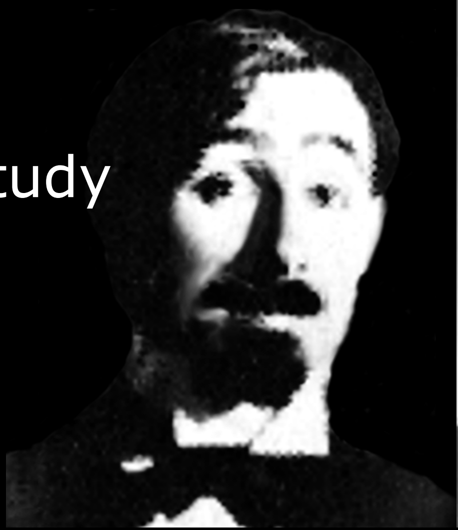
- Alonzo Church introduced the λ-calculus in the 1930s
  - His Motivation: study the mathematical foundations for functions
- Influence on language design
  - Anonymous functions in Scheme, Java, …
- Simple setting for studying types
  - Variants add types, quantifiers for polymorphism, etc.
- Mathematical model for language semantics
  - Dana Scott handled programs as data, as in ($\lambda x.$ (x x))

Combinatory Logic

Combinators predate the lambda calculus

- Two basic functions *K* and *S* characterized by
  - $K\ x\ y\ =\ x$
  - $S\ x\ y\ z = \ x\ z\ (\ y\ z\ )$

- Studied independently by Shönfinkel and Curry
  - Expressions built out of *K* and *S* are called *combinators.* Their study is called *combinatory logic*
  - "Astonishing conclusion that all functions having to do with the structure of functions can be built up out of" *K* and *S*"



Currying

Due to Gottlob Frege, but named after Haskell Curry

- Frege: functions of a single argument are enough
  - Given a function $f\,(x, y)$, we can define a function g that takes one argument at a time:

$$f\,(x, y) \;=\; g\;x\;y \;=\; (g\;x)\;y$$

  - In words, g is a function that maps one argument x to a function $(g\;x)$
  - Function $(g\;x)$ maps one argument y to $g\;x\;y = f\,(x, y)$

Syntactic Conventions

For the pure lambda calculus and variants

- Drop parentheses routinely; e.g.,

  MN for $(M\ N)$

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

- Function application associates to the left; e.g.,

  $M\ N\ P$ for $((M\ N)\ P)$

  $M\ (N\ P)$ for $(M\ (N\ P))$

- Use a single λ for a sequence of abstractions; e.g.,

  $\lambda xyz.\ M$ for $\lambda x.\ \lambda y.\ \lambda z.\ M$

Practice Problems

- Fully parenthesize the term
  - $x\ z\ (y\ z)$
- Write a lambda terms *I*, *K*, and *S, where*:
  - $I\ x\ = x$
  - $K\ x\ y = x$
  - $S\ x\ y\ z\ = \ x\ z\ (y\ z)$

Practice Problems

Solutions

- Fully parenthesize the term
  - $x\ z\ (y\ z) = ((x\ z)\ (y\ z))$
- A combinator is a lambda term without free variables. The three basic conbinators are:
  - $I = \lambda x.\ x$
  - $K = \lambda xy.\ x$
  - $S = \lambda xyz.\ x\ z\ (y\ z)$

Roadmap

Lambda Calculus: Basics

- Minimal Syntax
  - Variables, functions, function application
- Free and Bound Occurrences of Variables
  - Many have stumbled over the handling of scope issues
- Substitution of a term for a variable
  - Used for renaming and for symbolic computation
- Two rules for computation
  - Alpha Conversion (rnaming) & Beta Reduction (fun. application)

Roadmap

Lambda Calculus: Computation

- Convergence Property (Church-Rosser Theorem)
  - Each lambda term reduces to a unique normal form
- Computation Strategies
  - Counterparts of Call-By-Name and Coll-By-Value
- Computability (Church-Turing Thesis)
  - Any computation can be encoded in the lambd a calculus
  - Encodings for numbers, Booleans, functions, recursion

Free and Bound Variables

Informally, x is bound in $\lambda x.\ M$.

A variable is free if it is not bound.

Free and Bound Variables

- Give inductive rules for defining *free*(M), the set of free variables of the lambda term M
  - As a start,
  - $\quad free\,(x) \quad = \quad \{\, x \,\}$

- Recall the syntax of terms:

$$
\begin{array}{lll}
M \rightarrow & x & \textit{variables} \\
\quad\mid & (\, M\ N \,) & \textit{function application} \\
\quad\mid & (\, \lambda\, x.\ M \,) & \textit{abstraction}
\end{array}
$$

Rules for Free Variables

Free variables have been a trouble spot in both programming and in logic

- The set $free(M)$ is given by

$$
\begin{aligned}
free\ (x) &= \{\ x\ \} \\
free\ (M\ N) &= free\ (M) \cup free\ (N) \\
free\ (\lambda x.\ M) &= free\ (M) - \{\ x\ \}
\end{aligned}
$$

Free and Bound Occurrences

Example

- x has both free and bound occurrences in

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

- Notes
  – The first occurrence of x in this term is free
  – The occurrence of x in $(\lambda x.\ x\ y)$ is bound
  – There are no free occurrences of y

Free and Bound Occurrences

All occurrences of a variable are either free or bound

- Binding occurrence: *x* after lambda (as in λ*x*)
  - Convention: Abbreviate *binding occurrence* to simply *binding*
- Bound Occurrences of x in λ*x*. *M*
  - All free occurrence of *x* in *M* are *bound* by the binding λ*x*.*M*
  - The bound occurrences are in the *scope* of this binding
- Free occurrences
  - All unbound occurrences of a variable in a term are *free*

Preview of the Computation Rules

Alpha conversion and beta reduction motivate the careful treatment of Substitution

Alpha Conversion

Consistent renaming of bound variables

- Intuition
  - $\lambda x.x$ and $\lambda y.y$ represent the identity function
  - Changing a bound variable does not change the "value"
- α-*conversion* of $\lambda x.M$ *to* $\lambda y.M$
  - Idea: Substitute y for the free occurrences of x in M
  - Anything else? Any restrictions on y?
- α-equivalence of M and N
  - if M can be converted to N α-conversions

Alpha Conversion

Consistent renaming of bound variables

- Intuition
  - $\lambda x.x$ and $\lambda y.y$ represent the identity function
  - Changing a bound variable does not change the "value"
- α-*conversion* of $\lambda x.M$ *to* $\lambda y.M$
  - Idea: Substitute y for the free occurrences of x in M
  - Anything else?
  - y must not be free in M

Bbeta Reduction

Function application by substitution

- Notation
  - $[N/x]M$ is the result of substituting N for x in M
- β-Reduction: Intuition
  - $\lambda x.\ M$ is a function
  - $(\lambda x.\ M)\ N$ is the application of the function to N
  - So, reduce $(\lambda x.\ M)\ N$ to $[N/x]\ M$

Substitution

Give inductive rules for substitution

- As a start

$$[N/x]x \quad = \quad N$$

$$[N/x]y \quad = \quad y, \text{ if } x \neq y$$

Substitution

- What is the expected result of the following?

  $[\,(z\ y)\,/\,x\,]\,(\,\lambda y.\ x\ y\,)$

- What do your rules yield?

Capture of Free Variables

May need α-conversions before literal substitution

- Literal (incorrect) substitution
  - $[\ (z\ y)\ /\ x\ ]\ (\ \lambda y.\ x\ y\ ) \quad \neq \quad \lambda y.\ (z\ y)\ y$
- Why?
  - Replace $(\lambda y.\ x\ y)$ by the alpha-equivalent $(\lambda u.\ x\ u)$
  - Then, $[\ (z\ y)\ /\ x\ ](\ \lambda u.\ x\ u\ ) = \quad \lambda u.\ (z\ y)\ u$
- The issue
  - y is bound in $(\lambda y.\ x\ y)$ and free in $(z\ y)$
  - So, literal substitution results in the free y being captured by the binding λy

Substitution

Avoiding capture of free variables

$$
\begin{aligned}
[N/x]x &= N \\
[N/x]y &= y, \text{ if } x \neq y \\
[N/x]\,(M_1\; M_2) &= ([N/x]M_1)\;\; ([N/x]M_2) \\
[N/x]\,(\lambda x.M) &= \lambda x.M \\
[N/x]\,(\lambda y.M) &= \lambda y.\,[N/x](M), \text{ if } x \neq y \text{ and } y \notin \mathit{free}(N)
\end{aligned}
$$

Practice Problems

$[u / x]\ (x\ y)$ = ________________

$[u / x]\ (x\ u)$ = ________________

$[\lambda x.x / x]\ x$ = ________________

$[u / x]\ (y\ z)$ = ________________

$[u / x]\ (\lambda y.\ y)$ = ________________

$[\lambda x.x / x]\ y$ = ________________

$[u / x]\ (\lambda u.\ x)$ = ________________

$[u / x]\ (\lambda u.\ u)$ = ________________

Practice Problems

Solutions

$[u\,/\,x]\,(x\;y) \;=\; u\;y$

$[u\,/\,x]\,(x\;u) \;=\; u\;u$

No bound occurrences in M, so N replaces x

~~$[\lambda x.x\,/\,x]\;x \;=\; \lambda x.\,x$~~

$[u\,/\,x]\,(y\;z) \;=\; y\;z$

$[u\,/\,x]\,(\lambda y.\,y) \;=\; \lambda y.\,y$

No free occurrences of x in M, so no change

~~$[\lambda x.x\,/\,x]\;y \;=\; y$~~

$[u\,/\,x]\,(\lambda u.\,x) \;=\; [u\,/\,x]\,(\lambda z.\,x) \;=\; \lambda z.\,u$

$[u\,/\,x]\,(\lambda u.\,u) \;=\; [u\,/\,x]\,(\lambda z.\,z) \;=\; \lambda z.\,z$

Rules for Substitution

Avoiding capture of free variables

$$[N/x]x \quad = \quad N$$

$$[N/x]y \quad = \quad y, \;\; \text{if } x \neq y$$

$$[N/x]\,(M_1\ M_2) \quad = \quad ([N/x]M_1)\ \ ([N/x]M_2)$$

$$[N/x]\,(\lambda x.M) \quad = \quad \lambda x.M$$

$$[N/x]\,(\lambda y.M) \quad = \quad \lambda y.\ [N/x](M), \text{ if } x \neq y \text{ and } y \notin \mathit{free}(N)$$

Computation in the \-Calculus

Computation is symbolic, by reduction to a simpler term

Alpha Conversion and Beta Reduction

Notation and Terminology

- Beta Reduction Rule

$$(\lambda x.\ M)\ N \quad \Rightarrow_\beta \quad [N\ /\ x]\ M$$

  - In words, $(\lambda x.M)N$ beta-reduces to $[N/x]M$
  - $[N/x]M$ is considered simpler than $(\lambda x.M)N$
  - $(\lambda x.M)N$ is called a *redex*, from "reducible expression"
  - Notation: $\Rightarrow_\beta{}^*$ represents a sequence of zero or more beta reductions

- Alpha Conversion Rule

$$\lambda x.\ M \quad \Rightarrow_\alpha \quad \lambda y.\ [y\ /\ x]\ M \qquad \text{if } y \text{ is not free in } M$$

Normal Forms

Computations need not terminate

- Definition of Beta Normal Form
  - A term that cannot be beta reduced any further is said to be in β-*normal form* – often abbreviated to simply *normal form*
- Nonterminating Reductions
  - Example: the following reduction does not terminate

$$(\lambda x.\ xx)\ (\lambda x.\ xx) \Rightarrow_{\alpha} (\lambda y.\ yy)\ (\lambda x.\ xx)$$

$$\Rightarrow_{\beta} (\lambda x.\ xx)\ (\lambda x.\ xx)$$

Apply beta reductions as long as possible

$$SII = (\lambda \underline{xyz.\ xz\ (\ yz\ ))}\ I\ I$$

Note: $I\,z \;=\; z$

$$\downarrow$$

$$(\lambda yz.\ Iz\ (\ yz\ ))\ I$$

- Redexes are underlined

Alternative Reductions from *SII*

Redexes are underlined

Note: $I\,z \;=\; z$

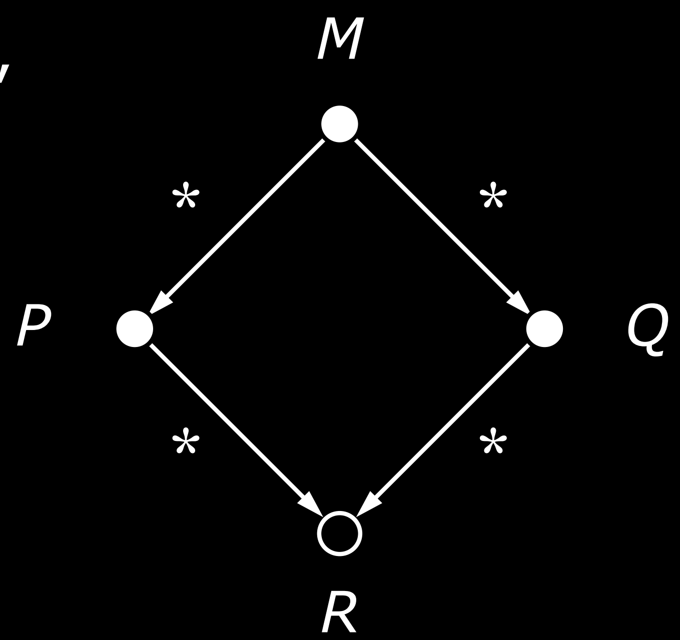
$$
\begin{tikzcd}[column sep=small]
 & SII = \underline{(\lambda xyz.\ xz\,(yz))}\ I\ I \arrow[d] & & \\
 & \underline{(\lambda yz.\ \underline{Iz}\,(yz))\ I} \arrow[dl] \arrow[dr] & & \\
\underline{(\lambda yz.\ z\,(yz))\ I} \arrow[dr] & & \lambda z.\ \underline{Iz}\,(\underline{Iz}) \arrow[dl] \arrow[dr] & \\
 & \lambda z.\ z\,(\underline{Iz}) \arrow[dr] & & \lambda z.\ \underline{Iz}z \arrow[dl] \\
 & & \lambda z.\ zz &
\end{tikzcd}
$$

$\lambda z.zz$ is in β-normal form

Convergence Property

Normal forms are unique, if they exist

- Church-Rosser Theorem
  - For all pure lambda-terms M, P, and Q,
  - if $M \Rightarrow_\beta^* P$ and $M \Rightarrow_\beta^* Q$,
  - then there must exist a term R
  - such that $P \Rightarrow_\beta^* R$ and $Q \Rightarrow_\beta^* R$



Reduction Strategies

Leftmost Outermost may terminate where Leftmost Innermost does not

Strategies for Choosing Redexes

Counterparts of Call-by-Name and Call-by-Value

- Reduction Strategy
  - For all terms *P* that are not in normal form, map P to *Q* such that $P \Rightarrow_\beta Q$
- Choose the Leftmost Outermost redex
  - Corresponds to Call-by-Name
- Choose the Leftmost Innermost redex
  - Corresponds to Call-by- Value

Identify the Leftmost-Outermost Reduction

$SII = (\lambda xyz.\ xz\ (\ yz\ ))\ I\ I$ Note: $I\ z\ =\ z$

$(\lambda yz.\ Iz\ (\ yz\ ))\ I$

$(\lambda yz.\ z\ (\ yz\ ))\ I$ $\lambda z.\ Iz\ (\ Iz\ )$

$\lambda z.\ z\ (\ Iz\ )$ $\lambda z.\ Izz$

$\lambda z.\ zz$

$\lambda z.zz$ is in β-normal form

Leftmost-Outermost Reduction

$SII = (\lambda \underline{xyz.\ xz\ (\ yz\ ))\ I}\ I$ Note: $I\ z = z$

$$\downarrow$$

$\underline{(\lambda yz.\ \underline{Iz}\ (\ yz\ ))\ I}$

$$\searrow$$

$\lambda z.\ \underline{Iz}\ (\ Iz\ )$

$$\swarrow$$

$\lambda z.\ z\ (\ \underline{Iz}\ )$

$$\searrow$$

$\lambda z.\ zz$

$\lambda z.zz$ is in β-normal form

Identify the Leftmost-Innermost Reduction

$SII = (\lambda xyz.\ \underline{xz\ (\ yz\ ))\ I}\ I$

Note: $I\ z\ =\ z$

$(\lambda yz.\ \underline{Iz}\ (\ yz\ ))\ I$

$(\lambda yz.\ z\ (\ yz\ ))\ I$

$\lambda z.\ \underline{Iz}\ (\ \underline{Iz}\ )$

$\lambda z.\ z\ (\ \underline{Iz}\ )$

$\lambda z.\ \underline{Iz}z$

$\lambda z.\ zz$

$\lambda z.zz$ **is in β-normal form**

Leftmost-Innermost Reduction

$SII = (\lambda \underline{xyz.\ xz\ (\ yz\ ))\ I}\ I$

Note: $I\ z = z$

$\downarrow$

$(\lambda yz.\ \underline{Iz}\ (\ yz\ ))\ I$

$\underline{(\lambda yz.\ z\ (\ yz\ ))\ I}$

$\lambda z.\ z\ (\ \underline{Iz}\ )$

$\lambda z.\ zz$

$\lambda z.zz$ is in β-normal form

Outermost Terminates, Innermost Doesn't

- Consider the combinators

$$I = \lambda\, x.\, x$$
$$K = \lambda\, x\, y.\, x$$
$$M = (\lambda x.\, xx)\ (\lambda x.\, xx)$$

- That is,

$$I\, x = x$$
$$K\, x\, y = x$$
$$M \textit{ does not terminate}$$

- Under Outermost,
  - evaluation of *KIM* terminates

$$K\, I\, M = I$$

- Under innermost,
  - evaluation of *KIM* does not terminate

Computation in the λ-Calculus

Church-Turing Thesis: Any computable function can be encoded in the lambda calculus

Computation in the Lambda Calculus

Church's encoding of the natural numbers

- Encode integers as lambda terms

0	$\lambda f.\ \lambda x.\ x$
1	$\lambda f.\ \lambda x.\ f\ x$
2	$\lambda f.\ \lambda x.\ f\ (f\ x)$
3	$\lambda f.\ \lambda x.\ f\ (f\ (f\ x))$

- Arithmetic functions

successor	$\lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x)$
plus	$\lambda m.\ \lambda n.\ m\ \mathit{successor}\ n$

Example: *successor* 2

Using Church's encoding of the natural numbers

$$
\begin{aligned}
\mathit{successor}\ 2 \quad &= \quad (\lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x))\ 2 \\
&\Rightarrow_\beta \quad \lambda f.\ \lambda x.\ f\ (2\ f\ x) \\
&= \quad \lambda f.\ \lambda x.\ f\ (\quad (\lambda g.\ \lambda y.\ g\ (g\ y)) \quad f\ x) \\
&\Rightarrow_\beta \quad \lambda f.\ \lambda x.\ f\ (\quad (\lambda y.\ f\ (f\ y)) \quad x) \\
&\Rightarrow_\beta \quad \lambda f.\ \lambda x.\ f\ (\ f\ (\ f\ x))) \\
&= \quad 3
\end{aligned}
$$

Computation in the Lambda Calculus

Other encodings

- Booleans

$$true = \lambda t.\ \lambda f.\ t$$

$$false = \lambda t.\ \lambda f.\ f$$

- Pairs

$$pair = \lambda x.\ \lambda y.\ \lambda f.\ f\ x\ y$$

$$first = \lambda p.\ p\ (true)$$

$$second = \lambda p.\ p\ (false)$$

Example: Reduction of *first* (*pair a b*)

- Given

$true \;=\; \lambda t.\ \lambda f.\ \ t$

$pair \;=\; \lambda x.\ \lambda y.\ \lambda f.\ \ f\,x$
y

$first \;=\; \lambda p.\ p\ (true)$

- Reduction

$first(pair\ ab)$n$= (\lambda p.p\ (true))\ (pair\ ab)$

$\Rightarrow_\beta\ (pair\ a\ b)\ (true)$

$=\ ((\lambda xyf.\ fxy)ab)\ (true)$

$\Rightarrow_\beta\ ((\lambda yf.\ \ fay)\ b)\ (true)$

$\Rightarrow_\beta\ (\lambda f.\ \ fab)\ (true)$

$\Rightarrow_\beta\ true\ a\ b$

$=\ (\lambda t\ f.\ t)\ a\ b$

$\Rightarrow_\beta\ (\lambda f.\ a)\ b$

$\Rightarrow_\beta\ a$

Fixed-Point Combinator

It's like recursion

- Fixed Point Equation
  - x is a fixed point of f if $x = f(x)$
- Fixed-Point Combinator
  - *fix* qualifies if or all functions f, $\mathit{fix}\ f \;=\; f\ (\ \mathit{fix}\ f\ )$
- Curry's Paradoxical Combinator
  - It's a specific fixed-point combinator:

$$Y \;=\; \lambda f.\ (\lambda x.\ \ f\ (x\ x))\ (\lambda x.\ \ f\ (x\ x))$$

Y is a fixed-point operator

$$
\begin{aligned}
Y\,g \;&=\; (\,\lambda f.\,(\lambda x.\;\, f\,(x\;x))\,(\lambda x.\;\, f\,(x\;x)))\;\; g \\
&=\; (\lambda x.\;\, g\,(x\;x))\,(\lambda x.\;\, g\,(x\;x)) \\
&=\; (\lambda y.\;\, g\,(y\;y))\,(\lambda x.\;\, g\,(x\;x)) \\
&=\; g\,((\lambda x.\;\, g\,(x\;x))\,(\lambda x.\;\, g\,(x\;x))) \\
&=\; g\,(\,Y\,g\,)
\end{aligned}
$$

References

– J. Barkley Rosser. Highlights of the history of the Lambda Calculus. *Annals of the History of Computing* 5, 4 (October 1984) 337-349.