Lambda Calculus

Reading: Scott, Ch. 11 on CD

Lecture Outline

- Lambda calculus, continued
  - Substitution, review
  - Rules of the lambda calculus
  - Normal forms
  - Reduction strategies

Syntax of Pure Lambda Calculus

Convention:
notation f, x, y, z for variables;
E, M, N, P, Q for expressions

- **$E ::= x \mid (\lambda x.\ E_1) \mid (E_1\ E_2)$**
  - A λ-expression is one of
    - Variable: **x**
    - Abstraction (i.e., function definition): **$\lambda x.\ E_1$**
    - Application: **$E_1\ E_2$**
- λ-calculus formulae (e.g., **(λx. (x y))**) are called expressions or terms
- **(λx. (x y))** corresponds to (lambda (x) (x y)) in Scheme!

Syntactic Conventions

- May drop parenthesis from $(E_1\ E_2)$ or $(\lambda x.\ E)$
  - E.g., $(f\ x)$ may be written as $f\ x$
- Function application is <u>left-associative</u>
  - I.e., it groups from left-to-right
  - E.g., $x\ y\ z$ abbreviates $((x\ y)\ z)$
  - E.g., $E_1\ E_2\ E_3\ E_4$ abbreviates $(((E_1\ E_2)\ E_3)\ E_4)$
- Application <u>has higher precedence</u> than abstraction
  - Another way to say this is that the scope of the **dot** extends as far to the right as possible
  - E.g., $\lambda x.\ x\ y = \lambda x.\ (x\ y) = (\lambda x.\ (x\ y)) \neq ((\lambda x.\ x)\ y)$

Free and Bound Variables

- Abstraction **(λx. E)** introduces a binding

- Variable **x** is said to be bound in **λx. E**
- The set of free variables of **E** is the set of variables that are unbound in **E**
- Defined by cases on **E**
  - Var **x**: free(**x**) = {**x**}
  - App $\mathbf{E_1}$ $\mathbf{E_2}$: free($\mathbf{E_1}$ $\mathbf{E_2}$) = free($\mathbf{E_1}$) U free($\mathbf{E_2}$)
  - Abs λ**x.E**: free(λ**x.E**) = free(**E**) - {**x**}

Substitution, formally

$(x\ y)[(\lambda v.v)/x] \rightarrow$
$(x[(\lambda v.v)/x]\ \ y[(\lambda v.v)/x])$
$\rightarrow ((\lambda v.v)\ y)$

- **(λx. E) M → E[M/x]** replaces all free occurrences of **x** in **E** by **M**
- **E[M/x]** is defined by cases on **E**:
  - Var: **y[M/x]** = **M** if **x = y**

    y[M/x] = **y** otherwise

    E.g. $x[(\lambda v.v)/x] \rightarrow \lambda v.v$
    $v[(\lambda v.v)/x] \rightarrow v$
  - App: **(E_1 E_2)[M/x]** = **(E_1[M/x] E_2[M/x])**
  - Abs: **(λy. E_1)[M/x]** = **λy. E_1** if **x = y**

    $(\lambda x.\ x x)[(\lambda v.v)/x] \rightarrow (\lambda x.\ x x)$

    (λy. E_1)[M/x] = **λz. ((E_1[z/y])[M/x])** otherwise,

    where **z** NOT in free(**E_1**) U free(**M**) U {**x**}

Substitution, formally

$1\_, 2\_, 3\_$ are fresh vars

$$(\lambda x.\overbrace{\lambda y.\ \lambda z.\ x\ z\ (y\ z)}^{E})\ \overbrace{v}^{M}$$

FUN!!!

$(\lambda y.\ E_1)[M/x] \Rightarrow$

$\lambda 2.\ ((E_1[2/y])[M/x])$

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

$\lambda 1\_.\ (((\lambda z.\ x\ z\ (y\ z))[1\_/y])[v/x])$

$(\lambda 2\_.\ (((x\ z\ (y\ z))[2\_/z])[1\_/y]))$

$(\lambda 2\_.\ x\ 2\_\ (1\_\ 2\_))[v/x]$

$(\lambda 3\_.\ (((x\ 2\_\ (1\_\ 2\_))[3\_/2\_])[v/x]))$

$\lambda 1\_.\ (\lambda 3\_.\ v\ 3\_\ (1\_\ 3\_))$

$\boxed{\lambda x.\ \lambda y.\ v\ y\ (x\ y)}$

Substitution, formally

$$(\lambda x.\underbrace{\lambda y.\ \lambda z.\ x\ z\ (y\ z)}_{E})\ \underbrace{v}_{M}$$

FUN!!!

$\lambda y.\ \lambda z.\ v\ z\ (y\ z)$

Rules (Axioms) of Lambda Calculus

- α rule (α-conversion): renaming of parameter (choice of parameter name does not matter)
  - $\lambda \mathbf{x.E} \rightarrow_\alpha \lambda \mathbf{z.(E[z/x])}$ provided that $\mathbf{z}$ is not free in $\mathbf{E}$
  - e.g., $\lambda \mathbf{x.\ x\ x}$ is the same as $\lambda \mathbf{z.\ z\ z}$

- β rule (β-reduction): function application (substitutes argument for parameter)
  - $(\lambda \mathbf{x.\ E})\ \mathbf{M} \rightarrow_\beta \mathbf{E[M/x]}$

  Note: **E[M/x]** as defined on previous slide!
  - e.g., $(\lambda \mathbf{x.\ x})\ \mathbf{z} \rightarrow_\beta \mathbf{z}$

Rules of Lambda Calculus: Exercises

- Use α-conversion and/or β-reduction:

$(\lambda x.\ x)\ y \rightarrow_{\alpha\beta}$? y

$(\lambda x.\ x)\ (\lambda y.\ y) \rightarrow_{\alpha\beta}$? λy.y

$(\lambda x.\lambda y.\lambda z.\ x\ z\ (y\ z))\ (\lambda u.\ u)\ (\lambda v.\ v) \rightarrow_{\alpha\beta}$

Notation: $\rightarrow_{\alpha\beta}$ denotes that expression on the left reduces to the expression on the right, through a sequence α-conversions and β-reductions.

Rules of Lambda Calculus: Exercises

- Use α-conversion or β-reduction:

$$(\lambda x.\lambda y.\lambda z.\ x\ z\ (y\ z))\ (\lambda u.\ u)\ (\lambda v.\ v) \rightarrow_{\alpha\beta}$$

E_1 E_2 E_3

$$(\lambda y.\lambda z.\ (\lambda u.u)\ z\ (y\ z))\ (\lambda v.v) \rightarrow$$

$$\lambda z.\ (\lambda u.u)\ z\ ((\lambda v.v)\ z) \rightarrow$$

$$\lambda z.\ z\ ((\lambda v.v)\ z) \rightarrow$$

$$\lambda z.\ z\ z$$

Reductions

- An expression **(λx.E) M** is called a redex (for reducible expression)
- An expression is in normal form if it cannot be β-reduced
- The normal form is the meaning of the term, the "answer"

Questions

- Is $\lambda z.\ z\ z$ in normal form? ✓
  - Answer: yes, it cannot be beta-reduced
- Is $(\lambda z.\ z\ z)\ (\lambda x.\ x)$ in normal form?
  - Answer: no, it can be beta-reduced

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

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

$\lambda x.x$

Lecture Outline

- Lambda calculus, continued
  - Substitution, review
  - Rules of the lambda calculus
  - Normal forms
  - Reduction strategies

Definitions of Normal Form

- **Normal form (NF)**: a term without redexes
- Head normal form (HNF) LAZY
  - **x** is in HNF
  - **(λx. E)** is in HNF if **E** is in HNF
  - **(x E_1 E_2 … E_n)** is in HNF
- Weak head normal form (WHNF)
  - **x** is in WHNF
  - **(λx. E)** is in WHNF
  - **(x E_1 E_2 … E_n)** is in WHNF

Questions

- λ**z. z z** is in NF, HNF, or WHNF?
- $\overbrace{(\lambda z.\ z\ z)}^{E_1}\ \overbrace{(\lambda x.\ x)}^{E_2}$ is in? Neither
- λ**x.**λ**y.**λ**z. x z (y (**λ**u. u))** is in? NF
- (We will be reducing to NF, mostly)

Questions

- $\overbrace{\mathbf{(\lambda x.\lambda y.\ x)}}^{E_1}\ \overbrace{\mathbf{z}}^{E_2}\ \overbrace{\mathbf{((\lambda x.\ z\ x)\ (\lambda x.\ z\ x))}}^{E_3}$ is in? Neither

Questions

- $\overbrace{\boxed{\mathbf{z}}}^{E_1}\ \overbrace{\mathbf{((\lambda x.\ z\ x)\ (\lambda x.\ z\ x))}}^{E_2}$ is in? in HNF, WHNF

Questions

$\lambda z.E$

not a NF
not a HNF

- λ**z.(**λ**x.**λ**y. x) z ((**λ**x. z x) (**λ**x. z x))** is in?

$\lambda z.\ (\lambda x.x)\ z$ is not a HNF because
$(\lambda x.x)\ z$ is not a HNF

it is a WHNF

More Reduction Exercises

- C = λx.λy.λf. f x y
- H = λf. f (λx.λy. x) T = λf. f (λx.λy. y)
- What is **H (C a b)**?

(λf. f (λx.λy.x)) (C a b)

(C a b) (λx.λy.x)

(λx.λy.λf. f x y) a b (λx.λy.x) →

(λy.λf. f a y) b (λx.λy.x) →

(λf. f a b) (λx.λy.x) → (λx.λy.x) a b →

(λy.a) b → a ✓

Exercise

An expression with no free variables is called combinator.
S, I, C, H, T are combinators.

- **S = λx.λy.λz. x z (y z)**
- **I = λx. x**
- What is **S I I I**?

Reducible expression is underlined at each step.

Lecture Outline

- Lambda calculus, continued
  - Substitution, review
  - Rules of the lambda calculus
  - Normal forms

  - Reduction strategies

Reduction Strategy

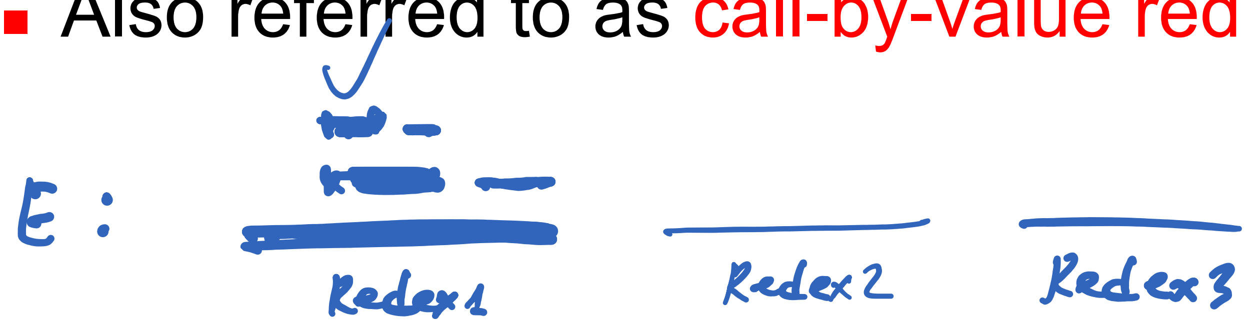
- Look again at **(λx.λy.λz. x z (y z)) (λu. u) (λv. v)**
- Actually, there are (at least) two "reduction paths":

Path 1: **<u>(λx.λy.λz. x z (y z)) (λu. u)</u> (λv. v)** $\rightarrow_\beta$
<u>(λy.λz. (λu. u) z (y z)) (λv. v)</u> $\rightarrow_\beta$
(λz. <u>(λu. u) z</u> ((λv. v) z)) $\rightarrow_\beta$ **(λz. z (<u>(λv. v) z</u>))** $\rightarrow_\beta$
λz. z z

Path 2: **<u>(λx.λy.λz. x z (y z)) (λu. u)</u> (λv. v)** $\rightarrow_\beta$
(λy.λz. <u>(λu. u) z</u> (y z)) (λv. v) $\rightarrow_\beta$
<u>(λy.λz. z (y z)) (λv. v)</u> $\rightarrow_\beta$ **(λz. z (<u>(λv. v) z</u>))** $\rightarrow_\beta$
λz. z z

Reduction Strategy

- A reduction strategy (also called evaluation order) is a strategy for choosing redexes
  - How do we arrive at a normal form (answer)?
- Applicative order reduction chooses the leftmost-innermost redex in an expression
  - Also referred to as call-by-value reduction



Reduction Strategy

- A reduction strategy (also called evaluation order) is a strategy for choosing redexes
  - How do we arrive at a normal form (answer)?
- Normal order reduction chooses the leftmost-outermost redex in an expression
  - Also referred to as call-by-name reduction

E : redex1 redex2 redex3

Reduction Strategy: Examples

- Evaluate **(λx. x x) ((λy. y) (λz. z))**
- Using applicative order reduction:

$(\lambda x. x x)(\lambda z. z) \rightarrow$

$(\lambda z. z)(\lambda z. z) \rightarrow \lambda z. z$

- Using normal order reduction

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

$((\lambda y. y)(\lambda z. z))((\lambda y. y)(\lambda z. z)) \rightarrow$

$(\lambda z. z)((\lambda y. y)(\lambda z. z)) \rightarrow (\lambda y. y)(\lambda z. z)$

$\rightarrow \lambda z. z$

Reduction Strategy

- In our examples, both strategies produced the same result. This is not always the case
  - First, look at expression **(λx. x x) (λx. x x)**. What happens when we apply β-reduction to this expression?
  - Then look at **(λz. y) ((λx. x x) (λx. x x))**
    - Applicative order reduction – what happens?
    - Normal order reduction – what happens?

Church-Rosser Theorem

- Normal form implies that there are no more reductions possible
- Church-Rosser Theorem, informally
  - If normal form exists, then it is unique (i.e., result of computation does not depend on the order that reductions are applied; i.e., no expression can have two distinct normal forms)
  - If normal form exists, then normal order will find it
- Church-Rosser Theorem, more formally:
  - For all pure λ-expressions **M**, **P** and **Q**, if **M** $\rightarrow^*$ **P** and **M** $\rightarrow^*$ **Q**, then there must exist an expression **R** such that **P** $\rightarrow^*$ **R** and **Q** $\rightarrow^*$ **R**

Reduction Strategy

- Intuitively:
- Applicative order (call-by-value) is an eager evaluation strategy. Also known as strict
- Normal order (call-by-name) is a lazy evaluation strategy
- What order of evaluation do most programming languages use?

Exercises

- Evaluate $(\lambda x.\lambda y.\ x\ y)\ ((\lambda z.\ z)\ w)$
- Using applicative order reduction
- Using normal order reduction

Exercise

- Let **S = λxyz. x z (y z)** and let **I = λx. x**
- Evaluate **S I I I** using applicative order

Exercise

- Let **S = λxyz. x z (y z)** and let **I = λx. x**
- Evaluate **S I I I** using normal order

The End