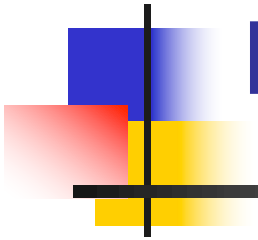
Announcements

- Rainbow grades
  - Homeworks 1 to 3, Quizzes 1 to 5 and Exam 1
  - If you have longstanding grading questions, please come see me in OH
- HW 5 is out, due Monday next week
- Exam 2 coming up next Friday



Lambda Calculus

Reading: Scott, Ch. 11.7 (on Companion Site)

Lecture Outline

- *Lambda calculus, continued*
  - *Free and bound variables, conclusion*
  - *Substitution*
  - *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 | (λx. E_1) | (E_1 E_2)**
  - A λ-expression is one of
    - Variable: **x**
    - Abstraction (i.e., function definition): **λx. E_1**
    - Application: **E_1 E_2**

- λ-calculus formulae (e.g., **(λx. (x y))**) are called expressions or terms

- **(λx. (x y))** just like (lambda (x) (x y)) in Scheme!

Syntactic Conventions

- May drop parenthesis from **(E_1 E_2)** or **(λx. E)**
  - E.g., **(f x)** may be written as **f x**
- Function application is left-associative
  - 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 has higher precedence than abstraction
  - Another way to say this is that the scope of the **dot** extends as far to the right as possible
  - E.g., **λx. x y** = **λx. (x y)** = **(λx. (x y))** ≠ **((λx. x) y)**

Free and Bound Variables

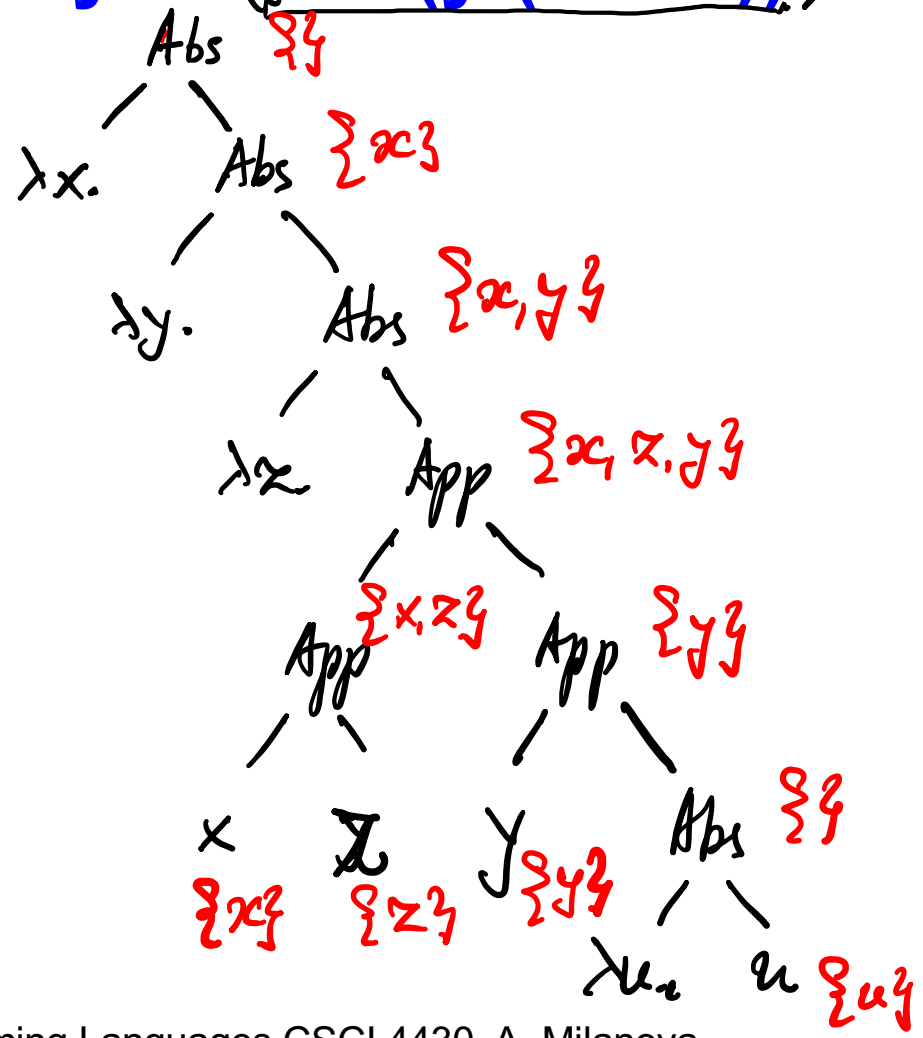
- Abstraction **(λx. E)** introduces a binding

- 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**}

Free and Bound Variables

((x z)(y (λu. u)))

- λx.λy.λz.((x z)(y (λu. u)))



Free and Bound Variables

- We must take free and bound variables into account when reducing expressions

  E.g., **(λx.λy. x y) (y w)**

  - Reduction rule defined in terms of substitution:

  (λy. x y) [(y w)/x]

  - First, rename bound **y** in **λy. x y** to **z**: **λz. x z**
    (more precisely, we have to rename to a variable that is NOT free in either **(y w)** or **(x y)**)
  - Second, replace **x** with argument **(y w)** safely:
    (λz. (y w) z) = λz. y w z

Substitution, formally

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

    $(\lambda y.\ E_1)[M/x]$ = **$\lambda z.\ ((E_1[z/y])[M/x])$** otherwise,

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

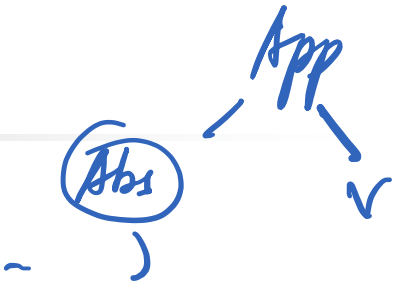
Substitution, formally

(λx.λy. x y) (y w)

- **(λy. x y)[(y w)/x]**
- **λa. (((x y)[a/y])[(y w)/x])**
- **λa. ((x a)[(y w)/x])**
- **λa. ((y w) a)**
- **λa. y w a**

Substitution, formally

$f(x) = x * x$ $f(5) = (x * x)[5/x]$



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

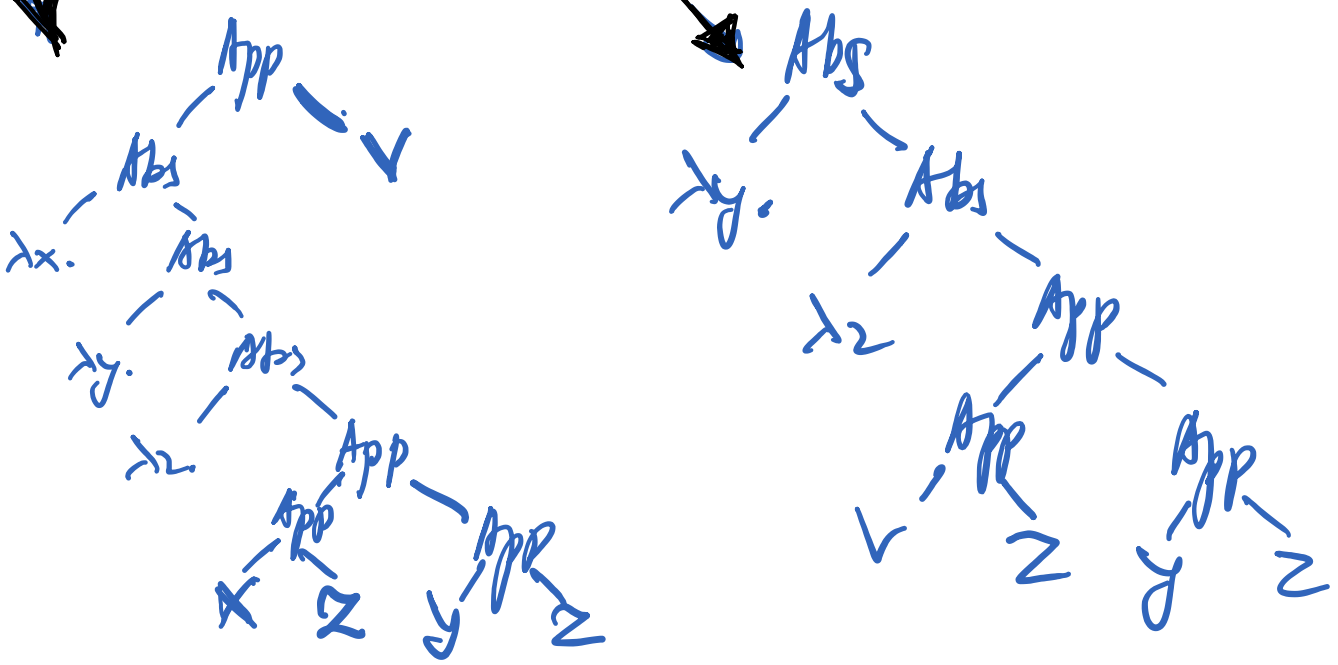
$(\lambda y.\lambda z.\ x\ z\ (y\ z))[v/x]$

Easy way: fn arg

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

Algorithmic substitution:

A lot longer



Substitution, formally

(λx.λy. λz. x z (y z)) v

Lecture Outline

- *Lambda calculus, continued*
  - *Free and bound variables, conclusion*
  - *Substitution*
  - *Rules of the lambda calculus*
  - *Normal forms*
  - *Reduction strategies*

Rules (Axioms) of Lambda Calculus

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

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

  $f(x) = x * x$ $f(5) \rightarrow (x*x)[5/x]$ $\rightarrow 5*5 \rightarrow 25$

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

    $x[z/x] \rightarrow z$

Rules of Lambda Calculus: Exercises

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

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

x E M

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

$E[M/x]$

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

$\lambda z.\ z\ z$

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 a sequence of β-reductions:

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

$(\lambda y.\lambda z.\ (\lambda u.u)\ z\ (y\ z))\ (\lambda v.\ v) \rightarrow_\beta$

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

$(\varepsilon_1\ \varepsilon_2)\ \varepsilon_3$

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

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

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

Reductions

x y

- An expression **(λx.E) M** is called a redex (for reducible expression)

E[M/x]

- An expression is in normal form if it cannot be β-reduced

- The normal form is the meaning of the term, the "answer"

Questions $(\lambda x.\ x\,x)(\lambda x.\ x\,x)$

- Is **λz. z z** in normal form?
  - Answer: yes, it cannot be beta-reduced

- Is **(λz. z z) (λx. x)** in normal form?
  - Answer: no, it can be beta-reduced

$(\lambda z.\ z\,z)(\lambda x.x) \rightarrow_\beta$

$(\lambda x.x)(\lambda x.x) \rightarrow_\beta$

$\lambda x.x$

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

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

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

$\lambda z.\ z\,z$

Lecture Outline

- *Lambda calculus, continued*
  - *Free and bound variables, conclusion*
  - *Substitution*
  - *Rules of the lambda calculus*
  - *Normal forms*
  - *Reduction strategies*

Definitions of Normal Form

- **Normal form (NF)**: a term without redexes
- Head normal form (HNF)
  - **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

- **λz. z z** is in NF, HNF, or WHNF?

NF => HNF => WHNF

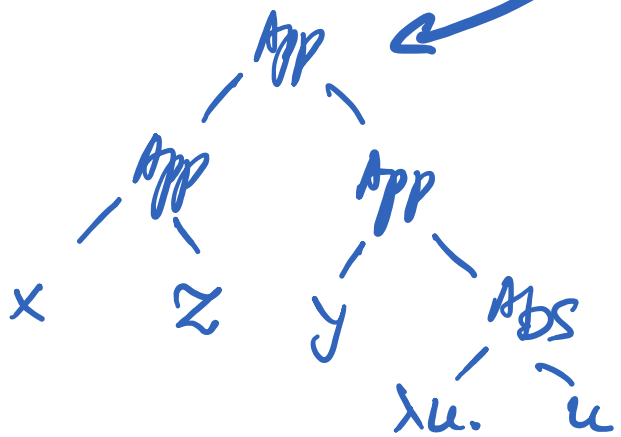
- **(λz. z z) (λx. x)** is in?

Neither

E_1 E_2

- **λx.λy.λz. x z (y (λu. u))** is in?

NF => HNF => WHNF



  - (We will be reducing to NF, mostly)

Questions

- **(λx.λy. x) z ((λx. z x) (λx. z x))** is in?

((E_1 E_2) E_3)

Not in NF.

Not in HNF.

Not in WHNF.

Questions

- **z ((λx. z x) (λx. z x))** is in?

Not in NF.
in WHNF. ✓

E1 E2

Questions

- **λz.(λx.λy. x) z ((λx. z x) (λx. z x))** is in?

Is in 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)**?

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*
  - *Free and bound variables, conclusion*
  - *Substitution*
  - *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:

Path 2:

Reduction Strategy

- Look again at **$(\lambda x.\lambda y.\lambda z.\ x\ z\ (y\ z))\ (\lambda u.\ u)\ (\lambda v.\ v)$**

- Actually, there are (at least) two "reduction paths":

Path 1: $(\lambda x.\lambda y.\lambda z.\ x\ z\ (y\ z))\ (\lambda u.\ u)\ (\lambda v.\ v) \rightarrow_\beta$
$(\lambda y.\lambda z.\ (\lambda u.\ u)\ z\ (y\ z))\ (\lambda v.\ v) \rightarrow_\beta$
$(\lambda z.\ (\lambda u.\ u)\ z\ ((\lambda v.\ v)\ z)) \rightarrow_\beta (\lambda z.\ z\ ((\lambda v.\ v)\ z)) \rightarrow_\beta$
$\lambda z.\ z\ z$

Path 2: $(\lambda x.\lambda y.\lambda z.\ x\ z\ (y\ z))\ (\lambda u.\ u)\ (\lambda v.\ v) \rightarrow_\beta$
$(\lambda y.\lambda z.\ (\lambda u.\ u)\ z\ (y\ z))\ (\lambda v.\ v) \rightarrow_\beta$
$(\lambda y.\lambda z.\ z\ (y\ z))\ (\lambda v.\ v) \rightarrow_\beta (\lambda z.\ z\ ((\lambda v.\ v)\ z)) \rightarrow_\beta$
$\lambda z.\ z\ z$

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

Reduction Strategy: Examples

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

- Using normal order reduction

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** →* **P** and **M** →* **Q**, then there must exist an expression **R** such that **P** →* **R** and **Q** →* **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 **(λx.λy. x y) ((λz. 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