Announcements

- Quiz 6 (at the end of class)

- HW5 due on Monday
- Exam 2 is next Friday
- Practice tests on Submitty

Lambda Calculus

Reading: Scott, Ch. 11.7 on Companion Website

Lecture Outline

- *Lambda calculus (catch-up)*
  - *Normal forms*
  - *Reduction strategies*

- *An applied lambda calculus*
- *The fixed-point operator*

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

- λ**z. z z** is in NF, HNF, or WHNF? NF

- $\underbrace{\textbf{(}\lambda\textbf{z. z z)}}_{E_1}\ \underbrace{\textbf{(}\lambda\textbf{x. x)}}_{E_2}$ 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

$\lambda x.x$

- $\underbrace{(\lambda x.\lambda y.\ x)}_{E_1}\ \underbrace{z}_{E_2}\ \underbrace{((\lambda x.\ z\ x)\ (\lambda x.\ z\ x))}_{E_3}$ is in? NEITHER

- $\underbrace{z}_{E_1}\ \underbrace{((\lambda x.\ z\ x)\ (\lambda x.\ z\ x))}_{E_2}$ is in? HNF, WHNF

- $\lambda z.(\lambda x.\lambda y.\ x)\ z\ ((\lambda x.\ z\ x)\ (\lambda x.\ z\ x))$ is in?

  WHNF

More Reduction Exercises

$(\lambda x.E)\ M$

COMBINATORS

- **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)**? ⤳ mean fup is $\boxed{a}$

$= (\lambda f. f (\lambda x.\lambda y. x))\ (C\ a\ b)$

$(C\ a\ b)\ (\lambda x.\lambda y. x) =$

$((\lambda x.\lambda y.\lambda f. f\ x\ y)\ a\ b)\ (\lambda x.\lambda y. x) \to_\beta$

$((\lambda y.\lambda f. f\ a\ y)\ b)\ (\lambda x.\lambda y. x) \to_\beta$

$(\lambda f. f\ a\ b)\ (\lambda x.\lambda y. x) \to_\beta (\lambda x.\lambda y. x)\ a\ b \to_\beta$

$(\lambda y. a)\ b \to_\beta \boxed{a}$

Exercise

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

$(\lambda x.E)\ M$

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

$\rightarrow I = \lambda x.x$

Reducible expression is underlined at each step.

$$\underline{(\lambda x.\lambda y.\lambda z.\ x\ z\ (y\ z))\ I}\ I\ I \rightarrow_\beta$$

$$\underline{(\lambda y.\lambda z.\ I\ z\ (y\ z))\ I}\ I \rightarrow_\beta$$

$$\underline{(\lambda z.\ I\ z\ (I\ z))\ I} \rightarrow_\beta \underline{I\ I}\ (I\ I) = \underline{(\lambda x.x)\ I}\ (I\ I)$$

$$\rightarrow_\beta \underline{I\ (I\ I)} = \underline{(\lambda x.x)\ (I\ I)} \rightarrow_\beta \underline{I\ I} = (\lambda x.x)\ I \rightarrow_\beta$$

$$I = \lambda x.x$$

Reduction Strategy

- Look again at $\underbrace{\mathbf{(\lambda x.\lambda y.\lambda z.\ x\ z\ (y\ z))}}_{S}\ \underbrace{\mathbf{(\lambda u.\ u)}}_{I}\ \underbrace{\mathbf{(\lambda v.\ v)}}_{I}$

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

- There are two (actually, more) "reduction paths":

Path 1: $\underline{(\lambda y.\lambda z.(\lambda u.u)\ z\ (y\ z))\ (\lambda v.v)} \to_\beta$

$\lambda z.\ \underline{(\lambda u.u)\ z}\ ((\lambda v.v)\ z) \to_\beta \lambda z.\ z\ (\underline{(\lambda v.v)\ z}) \to_\beta$

$\lambda z.\ z\ z$ NF

Path 2: $(\lambda y.\lambda z.\ \underline{(\lambda u.u)\ z}\ (y\ z))\ (\lambda v.v) \to_\beta$

$\underline{(\lambda y.\lambda z.\ z\ (y\ z))\ (\lambda v.v)} \to_\beta$

$\lambda z.\ z\ (\underline{(\lambda v.v)\ z}) \to_\beta \lambda z.\ z\ z$ NF

Reduction Strategy

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

- There are two (actually, more) "reduction paths":

Path 1: $\underline{(\lambda x.\lambda y.\lambda z.\ x\ z\ (y\ z))\ (\lambda u.\ u)}\ (\lambda v.\ v) \to_\beta$

$\underline{(\lambda y.\lambda z.\ (\lambda u.\ u)\ z\ (y\ z))\ (\lambda v.\ v)} \to_\beta$

$(\lambda z.\ \underline{(\lambda u.\ u)\ z}\ ((\lambda v.\ v)\ z)) \to_\beta (\lambda z.\ z\ (\underline{(\lambda v.\ v)\ z})) \to_\beta$

$\lambda z.\ z\ z$

Path 2: $\underline{(\lambda x.\lambda y.\lambda z.\ x\ z\ (y\ z))\ (\lambda u.\ u)}\ (\lambda v.\ v) \to_\beta$

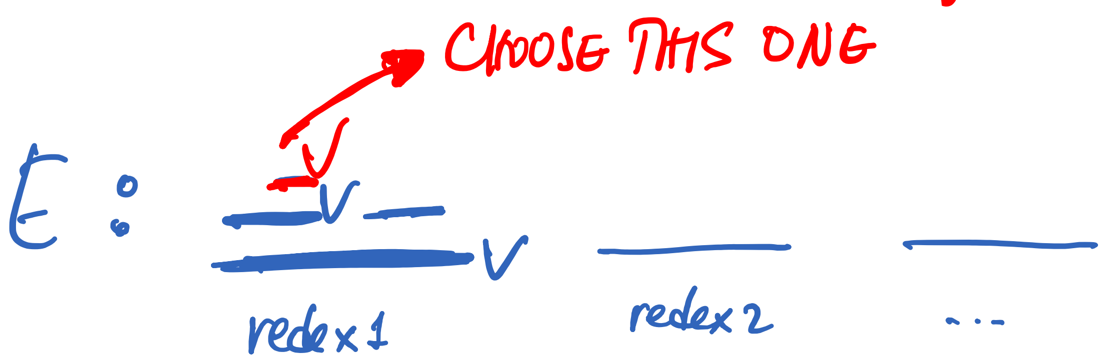
$(\lambda y.\lambda z.\ \underline{(\lambda u.\ u)\ z}\ (y\ z))\ (\lambda v.\ v) \to_\beta$

$\underline{(\lambda y.\lambda z.\ z\ (y\ z))\ (\lambda v.\ v)} \to_\beta (\lambda z.\ z\ (\underline{(\lambda v.\ v)\ z})) \to_\beta$

$\lambda z.\ z\ z$

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

CHOOSE THIS ONE

E : ______ ✓ ______ ______

redex 1 redex2

Reduction Strategy: Examples

$(\lambda x. E)\ M$

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

$(\lambda x. x\, x)\ (\underline{(\lambda y. y)(\lambda z. z)}) \rightarrow_\beta \underline{(\lambda x. x\, x)\ (\lambda z. z)} \rightarrow_\beta$

$\underline{(\lambda z. z)\ (\lambda z. z)} \rightarrow_\beta \lambda z. z$

- Using normal order reduction

$\underline{((\lambda y. y)\ (\lambda z. z))}\ ((\lambda y. y)(\lambda z. z)) \rightarrow_\beta$

$\underline{(\lambda z. z)\ ((\lambda y. y)(\lambda z. z))} \rightarrow_\beta \underline{(\lambda y. y)(\lambda z. z)} \rightarrow_\beta$

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

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 **(λx.λy. y) ((λx. x x) (λx. x x)) z**
    - Applicative order reduction – what happens?
    - Normal order reduction – what happens?

$(\lambda x.\lambda y.y)((\lambda x.xx)(\lambda x.xx))\ z \rightarrow (\lambda y.y)\ z \rightarrow_\beta z$

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: $(e_0 \; e_1 \; e_2 \; \ldots \; e_n)$
- 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

$(\lambda x. \lambda y. x\, y)\ w \to_\beta \lambda y. w\, y$

- Using normal order reduction

$\lambda y. ((\lambda z. z)\ w)\ y \to_\beta \lambda y. w\, y$

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

Lecture Outline

- *Lambda calculus (catch-up)*
  - *Normal forms*
  - *Reduction strategies*

- *An applied lambda calculus and*
- *The fixed-point operator*

Y combinator:

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

Applied Lambda Calculus (from Sethi)

- **E ::= c | x | (λx.E$_1$) | (E$_1$ E$_2$)**

  An applied lambda calculus augments the pure lambda calculus with constants. It defines its set of constants and reduction rules. For example:

Constants:

if, true, false

(all these are λ terms,

e.g., true=λ**x.**λ**y. x!**)

0, iszero, pred, succ

Reduction rules:

if true M N $\rightarrow_\delta$ M

if false M N $\rightarrow_\delta$ N

iszero 0 $\rightarrow_\delta$ true

iszero (succ^k 0) $\rightarrow_\delta$ false, k>0

iszero (pred^k 0) $\rightarrow_\delta$ false, k>0

succ (pred M) $\rightarrow_\delta$ M

pred (succ M) $\rightarrow_\delta$ M

From an Applied Lambda Calculus to a Functional Language

Construct	Applied λ-Calculus	A Language (ML)	
Variable	x	x	x in Scheme
Constant	c	c	c
Application	M N	M N	(M N)
Abstraction	λx.M	fun x => M	(lambda (x) M)
Integer	$succ^k$ 0, k>0	k	
	$pred^k$ 0, k>0	-k	
Conditional	if P M N	if P then M else N	(if P M N)
Let	(λx.M) N	let val x = N in M end	(let ((x N)) M)

The Fixed-Point Operator

- One more constant, and one more rule:

  fix $\qquad$ **fix M** $\rightarrow_\delta$ **M (fix M)**

  $\boxed{\textbf{M(M(M...M(fix M)...))}}$

- Needed to define recursive functions:

$$\underline{\textbf{plus}}\ \textbf{x y} = \begin{cases} \textbf{y} & \text{if } \textbf{x = 0} \\ \underline{\textbf{plus}}\ \underbrace{\textbf{(pred x)}}_{\boxed{\text{x-1}}}\ \underbrace{\textbf{(succ y)}}_{\boxed{\text{y+1}}} & \text{otherwise} \end{cases}$$

- Therefore:

$\underline{\textbf{plus}} = \lambda$**x.**$\lambda$**y. if (iszero x) y (**$\underline{\textbf{plus}}$ **(pred x) (succ y))**

The Fixed-Point Operator

- But how do we define **plus**?

plus 2 3 $\to_\delta$ plus 1 4 $\to$

Define **plus** = **fix M**, where

M = λ**f**. λ**x**.λ**y**. **if** (**iszero x**) **y** (**f** (**pred x**) (**succ y**))

$$(\text{fix } M)\ x\ y = \begin{cases} y & \text{if } x \text{ is } 0 \\ (\text{fix } M)\ (\text{pred } x)\ (\text{succ } y) & \end{cases}$$

We must show that

fix M $\to_{\delta\beta}$?

λ**x**.λ**y**. **if** (**iszero x**) **y** ((**fix M**) (**pred x**) (**succ y**))

(x-1) (y+1)

plus = fix M

ONLY WORKS WITH NORMAL ORDER!

$(\underbrace{\text{fix } M}_{\text{plus}}) \longrightarrow_{\delta}$

$M\ (\text{fix } M) =$

$$\underline{(\lambda f.\lambda x.\lambda y.\text{ if } (\text{iszero } x)\ y\ (f\ (\text{pred } x)\ (\text{succ } y)))\ (\text{fix } M)} \longrightarrow_{\beta}$$

$$\lambda x.\lambda y.\text{ if } (\text{iszero } x)\ y\ ((\underbrace{\text{fix } M}_{\text{plus}})\ (\text{pred } x)\ (\text{succ } y))$$

$$(\text{fix } M)\ 2\ 3 \Rightarrow \underline{(\lambda x.\lambda y.\text{ if } (\text{iszero } x)\ y\ ((\text{fix } M)\ (\text{pred } x)\ (\text{succ } y)))\ 2\ 3}$$

$$\longrightarrow_{\beta}^{*} \text{ if } (\underline{\text{iszero } 2})\ 3\ ((\text{fix } M)\ (2-1)\ (3+1)) \longrightarrow_{\beta}^{*} (\text{fix } M)\ (2-1)\ (3+1)$$

$$\longrightarrow_{\beta}^{*} \ldots\ 3+1+1 = \underline{5}$$

The Fixed-Point Operator

We have to show NORMAL ORDER!

$(\mathbf{fix}\ M) \to_{\delta\beta}$

$\lambda x.\lambda y.\ \mathbf{if}\ (\mathbf{iszero}\ x)\ y\ ((\mathbf{fix}\ M)\ (\mathbf{pred}\ x)\ (\mathbf{succ}\ y))$

$(\mathbf{fix}\ M) \to_{\delta} M\ (\ \mathbf{fix}\ M\ ) =$

$(\lambda f.\ \lambda x.\lambda y.\ \mathbf{if}\ (\mathbf{iszero}\ x)\ y\ (f\ (\mathbf{pred}\ x)\ (\mathbf{succ}\ y)))\ (\ \mathbf{fix}\ M\ ) \to_{\beta}$

$\lambda x.\lambda y.\ \mathbf{if}\ (\mathbf{iszero}\ x)\ y\ ((\mathbf{fix}\ M)\ (\mathbf{pred}\ x)\ (\mathbf{succ}\ y))$

The Fixed-Point Operator

Define **times** =

fix (λf.λx.λy. if (iszero x) 0 (plus y (f (pred x) y)))

Exercise: define **factorial** = ?

The Y Combinator

Property of Fixed-Point Operator:

$fix\ M \to_\beta^* M(fix\ M)$

- **fix** is, of course, a lambda expression!
- One possibility, the famous Y combinator:

Y = λf. (λx. f (x x)) (λx. f (x x))

WHNF. NORMAL ORDER

fix = Y

Show that **Y M** indeed reduces to **M (Y M)**

$YM = (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))\ M \to_\beta$

$(\lambda x.M(xx))(\lambda x.M(xx)) \to_\beta$

$M\ ((\lambda x.M(xx))(\lambda x.M(xx))) = M(YM)$

YM

YM

The End