Lambda Calculus (3A) - Reduction

Young Won Lim
9/26/22

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Beta reduction (1)

A <u>function application</u> **λx.e f** is <u>evaluated</u>

by substituting the **argument f**

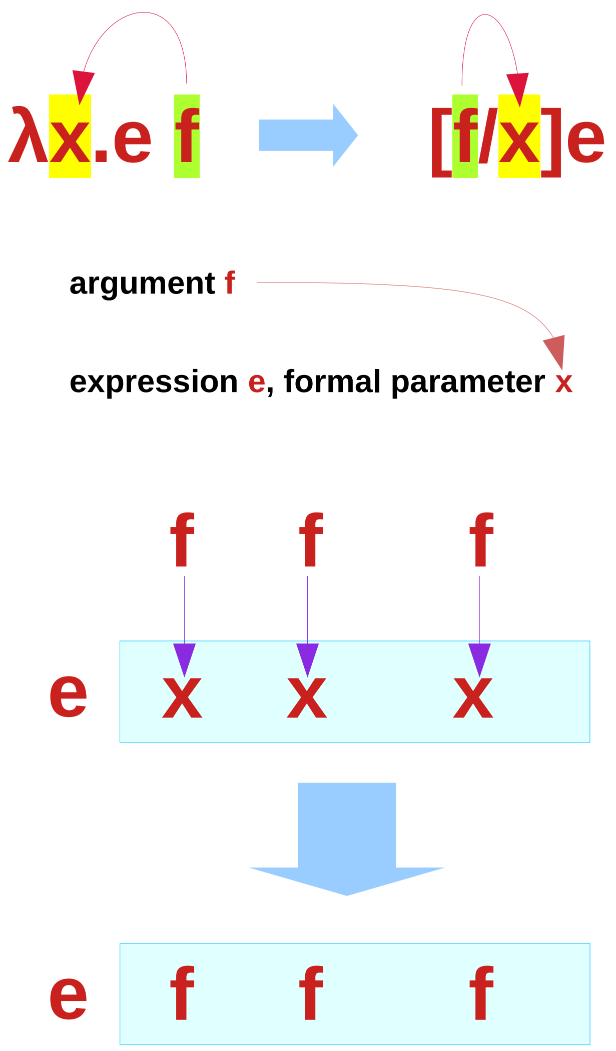
for *all free occurrences* of the **formal parameter x**

in the body **e** of the **function definition**.

We will use the notation **[f/x]e** to indicate

that **f** is to be <u>substituted</u> for *all free occurrences* of **x**

in the expression **e**.



http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Beta reduction (2)

Examples:

(λx.x)y → [y/x]x = y

in the express **x**,
substitute the parameter **x** with the argument **x**

(λx.xzx)y → [y/x]xzx = yzy

in the express **xzx**,
substitute the parameter **x** with the argument **y**

(λx.z)y → [y/x]z = z

in the express **z**,
substitute the parameter **x** with the argument **y**

since the formal parameter **x** does not appear in the body **z**.

This **substitution** in a **function application** is called
a **beta reduction** and we use a right arrow to indicate it.

λx.e f → [f/x]e

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Beta reduction (3)

If **expr1** → **expr2**, we say **expr1** reduces to **expr2** in one step.

In general, **(λx.e)f** → **[f/x]e** means that

applying the **function (λx.e)** to the **argument expression f**
reduces to the **expression [f/x]e**

where the **argument expression f** is substituted
for the function's **formal parameter x** in the **function body e**.

λx.e f → **[f/x]e**

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Beta reduction (4)

A **lambda calculus expression** (aka a "**program**") is
"run" by *computing a final result*
by repeatly applying **beta reductions**.

We use →* to denote the reflexive and transitive closure of →;
that is, zero or more applications of **beta reductions**.

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Beta reduction (5)

Examples:

(λx.x)y → y

illustrating that **λx.x** is the **identity function**

(λx.xx)(λy.y) → **(λy.y)(λy.y)** → **(λy.y)**;

thus, we can write **(λx.xx)(λy.y) →* (λy.y)**.

we have <u>applied</u> a **function** to a **function**

as an argument and the **result** is a **function**.

λx.e f ⇒ **[f/x]e**

(λx.xx) (λy.y) **function argument**

(λy.y)(λy.y) **function application**

(λy.y)(λy.y) **indentity function**

(λy.y)

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Beta reduction (6)

Examples:

(λx.x)y → y

illustrating that **λx.x** is the **identity function**

(λx.xx)(λy.y) → (λy.y)(λy.y) → (λy.y);

thus, we can write (λx.xx)(λy.y) →* (λy.y).

→* to denote the reflexive and transitive closure of →

that is, zero or more applications of beta reductions

(λx.xxx)(λy.y) → (λy.y)(λy.y)(λy.y) → (λy.y) (λy.y) → (λy.y) ;

Transitive relation

x R y and **y R z** then **x R z**

Reflexive relation

x R x

(λx.x)	→	(λy.y)
(λx.xx)(λy.y)	→	(λy.y)
(λx.xxx)(λy.y)	→	(λy.y) ;

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Beta reduction (6)

(λx.xxx)(λy.y) → ((λy.y)(λy.y))(λy.y)

→ (λy.y) (λy.y) ;

→ (λy.y) ;

Transitive relation

x R y and **y R z** then **x R z**

Reflexive relation

x R x

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Beta reduction (7)

Evaluation of a **lambda abstraction** (beta-reduction)

is just substitution:

(λx . + x 1) 4 → (+ 4 1) → 5

The argument may appear more than once

(λx . + x x) 4 → (+ 4 4) → 8

or not at all

(λx . 3) 5 → 3

http://www.cs.columbia.edu/~aho/cs4115/Lectures/2014_EdwardsLC.pdf

Beta reduction (8)

extra parentheses may help.

(λx . λy . + x y) 3 4 = ((λx . (λy . ((+ x) y))) 3) 4
→ (λy . ((+ 3) y)) 4
→ ((+ 3) 4)
→ 7

functions may be arguments

(λ f . f 3) (λx . + x 1) → (λx . + x 1) 3
→ (+ 3 1)
→ 4

(λx . λy . + x y) 3 4
= (λx . λy . (+ x) y) 3 4
= (λx . λy . ((+ x) y)) 3 4
= (λx . (λy . ((+ x) y))) 3 4
= ((λx . (λy . ((+ x) y))) 3) 4

(λx . λy . + x y) 3 4
= ((λx . λy . + x y) 3) 4
= ((λx . (λy . + x y)) 3) 4
= ((λx . (λy . (+ x y))) 3) 4
= ((λx . (λy . ((+ x) y))) 3) 4

http://www.cs.columbia.edu/~aho/cs4115/Lectures/2014_EdwardsLC.pdf

Beta reduction (9)

(λx . + x y) 4

Here, **x** is like a function argument
but **y** is like a global variable.

Technically, **x** occurs bound
and **y** occurs free in
(λx . + x y)

However, both **x** and **y** occur free in
(+ x y)

http://www.cs.columbia.edu/~aho/cs4115/Lectures/2014_EdwardsLC.pdf

Beta reduction (10)

(λx . E) F $\rightarrow_\beta$ **E ′**

E ′ is obtained from **E**

by replacing every instance of **x**

that appears free in **E**

with **F** .

http://www.cs.columbia.edu/~aho/cs4115/Lectures/2014_EdwardsLC.pdf

Beta reduction (11)

The definition of free and bound mean

variables have scopes.

(λx . + (− x 1)) x 3

here, the rightmost **x** appears free

(λx . (λx . + (− x 1)) x 3) 9 → **(λ x . + (− x 1)) 9 3**

→ **+ (− 9 1) 3**

→ **+ 8 3**

→ **11**

http://www.cs.columbia.edu/~aho/cs4115/Lectures/2014_EdwardsLC.pdf

Beta reduction (12)

Another Example

(λx . λy . + x ((λx . − x 3) y)) 5 6

→ (λy . + 5 ((λx . − x 3) y)) 6

→ + 5 ((λx . − x 3) 6)

→ + 5 (− 6 3)

→ + 5 3

→ 8

((λx . λy . + x ((λx . − x 3) y)) 5) 6

→ (λy . + 5 ((λx . − x 3) y)) 6

→ + 5 ((λx . − x 3) 6)

→ + 5 (− 6 3)

→ + 5 3

→ 8

http://www.cs.columbia.edu/~aho/cs4115/Lectures/2014_EdwardsLC.pdf

Explicit parenthesis

1. Lambda calculus

Make all parentheses explicit in the following λ-expressions

a. λx.xz λy.xy

(λx.((x z) (λy.(x y))))

b. (λx.xz) λy.w λw.wyzx

(λx.xz) (λy.w λw.wyzx)

(λx.xz) (λy.w λw.wyzx)

((λx.(x z)) (λy.(w (λw.((((w y) z) x))))))

c. λx.xy λx.yx

(λx.((x y) (λx.(y x))))

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Unbound variables

Find all free (unbound) variables in the following λ-expressions

d. λx.x z λy.x y

(λx.((x z) (λy.(x y))))

e. (λx. x z) λy. w λw. w y z x

((λx.(x z)) (λy.(w (λw.((((w y) z) x)))))

f. λx. x y λx. y x

(λx.((x y) (λx.(y x))))

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Beta reduction

Apply β-reduction to the following λ-expressions as much as possible

g. (λz.z) (λy.y y) (λx.x a) // β-reduction = body[sym/replacement]

(λz.z) (λy.y y) (λx.x a) // z [z/(λy.y y)] replace z with λy.y y

(λy.y y) (λx.x a) // y y[y/(λx.x a)] replace y with λx.x a

(λx.x a) (λx.x a) // x a[x/(λx.x a)] replace x with λx.x a

(λx.x a) a // x a[x/a] replace x with a

a a

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Applying beta reduction (1)

Apply β-reduction to the following λ-expressions as much as possible

h. (λz.z) (λz.z z) (λz.z y)

(λz.z) (λz.z z) (λz.z y) // β-reduction: replace z with λz.z z

(λz.z z) (λz.z y) // β-reduction: replace z with λz.z y

(λz.z y) (λz.z y) // β-reduction: replace z with λz.z y

(λz.z y) y // β-reduction: replace z with y

y y

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Applying beta reduction (2)

i. (λx.λy.x y y) (λa.a) b

(λx.λy.x y y) (λa.a) b // β-reduction: replace x with λa.a

(λy.(λa.a) y y) b // β-reduction: replace y with b

(λa.a) b b // β-reduction: replace a with b

b b

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Applying beta reduction (3)

j. (λx.λy.x y y) (λy.y) y

(λx.λy.x y y) (λy.y) y // α-conversion: rename y to a

(λx.λa.x a a) (λy.y) y // β-reduction: replacing x with λy.y

(λa.(λy.y) a a) y // β-reduction: replacing a with y

(λy.y) y y // β-reduction: replacing y with y

y y

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Applying beta reduction (4)

k. (λx.x x) (λy.y x) z

(λx.x x) (λy.y x) z // β-reduction: replacing x with λy.y x

(λy.y x) (λy.y x) z // β-reduction: replacing y with λy.y x

(λy.y x) x z // β-reduction: replacing y with x

x x z

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Applying beta reduction (5)

I. (λx. (λy. (x y)) y) z

(λx. (λy. (x y)) y) z // α-conversion: rename y to a

(λx. (λa. (x a)) y) z // β-reduction: replacing x with z

(λa. (z a)) y // β-reduction: replacing a with y

z y

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Applying beta reduction (6)

m. ((λx.x x) (λy.y)) (λy.y)

((λx.x x) (λy.y)) (λy.y) // β-reduction: replacing x with λy.y

((λy.y) (λy.y)) (λy.y) // β-reduction: replacing y with λy.y

(λy.y) (λy.y) // β-reduction: replacing y with λy.y

λy.y

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Applying beta reduction (7)

n. (((λx. λy.(x y))(λy.y)) w)

(((λx. λy.(x y))(λy.y)) w)	// α-conversion: rename y to a
(((λx. λa.(x a))(λy.y)) w)	// β-reduction: replacing x with λy.y
((λa.((λy.y) a)) w)	// β-reduction: replacing a with w
(λy.y) w	// β-reduction: replacing y with λy.y
w	

(((λx. λy.(x y))(λy.y)) w)	// α-conversion: rename y to a
(((λx. λa.(x a))(λy.y)) w)	// β-reduction: replacing x with λy.y
((λa.((λy.y) a)) w)	// β-reduction: replacing y with a
((λa. a)) w)	// β-reduction: replacing a with w
w	

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Multiple reduction seuqneces

multiple reduction sequences

o. (λx.y) ((λy.y y y) (λx.x x x))

// β-reduction: replace x in λx.y with ((λy.y y y) (λx.x x x))

(λx.y) ((λy.y y y) (λx.x x x))

// no x in body, so just discard argument

~~((λy.y y y) (λx.x x x))~~

// and replace (λx.y) <...> with y

y

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

Multiple reduction seuqneces

multiple reduction sequences

o. (λx.y) ((λy.y y y) (λx.x x x))

OR

// β-reduction: replace y in λy.y y y with λx.x x x

(λx.y) ((λy.y y y) (λx.x x x))

// Can repeat β-reduction for x as many times as we wish!

(λx.y) ((λx.x x x) (λx.x x x) (λx.x x x) (λx.x x x))

(λx.y) ((λx.x x x) (λx.x x x) (λx.x x x) (λx.x x x) (λx.x x x))

(λx.y) ((λx.x x x) (λx.x x x) (λx.x x x) (λx.x x x) (λx.x x x) (λx.x x x))

https://www.cs.umd.edu/class/fall2017/cmsc330/tests/prac8-soln-fall13.pdf

- CFG for the Lambda Calculus
- Function Abstraction
- Function Application
- Free and Bound Variables
- Beta Reductions
- **Evaluating a Lambda Expression**
- Currying
- Renaming Bound Variables by Alpha Reduction
- Eta Conversion
- Substitutions
- Disambiguating Lambda Expressions
- Normal Form
- Evaluation Strategies

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Evaluating a Lambda expression (1)

A **lambda calculus expression** can be thought of
as a **program** which can be executed by evaluating it.

Evaluation is performed by *repeatedly* finding
a **reducible expression** (called a **redex**)
and reducing it by a **function evaluation**
until there are no more **redexes**.

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Evaluating a Lambda expression (2)

Example 1:

The lambda expression **(λx.x)y** in its entirety is

a **redex** that reduces to **y**.

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Evaluating a Lambda expression (3)

Example 2:

The lambda expression **(+ (* 1 2) (- 4 3))** has

two **redexes**: **(* 1 2)** and **(- 4 3)**.

If we choose to reduce the *first* **redex**, we get **(+ 2 (- 4 3))**.

We can then reduce **(+ 2 (- 4 3))** to get **(+ 2 1)**.

Finally we can reduce **(+ 2 1)** to get **3**.

Note that if we had chosen the *second* **redex** to evaluate first,

we would have ended up with the same result:

(+ (* 1 2) (- 4 3)) → **(+ (* 1 2) 1)** → **(+ 2 1)** → **3.**

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

- CFG for the Lambda Calculus
- Function Abstraction
- Function Application
- Free and Bound Variables
- Beta Reductions
- Evaluating a Lambda Expression
- Currying
- Renaming Bound Variables by Alpha Reduction
- **Eta Conversion**
- Substitutions
- Disambiguating Lambda Expressions
- Normal Form
- Evaluation Strategies

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Eta conversion

The two **lambda expressions**

(λx.+ 1 x) and **(+ 1)**

are equivalent in the sense that

these expressions *behave* in exactly the same way

when they are applied to an **argument**

-- they add 1 to it.

η-conversion is a rule that **expresses** this **equivalence**.

In general, if **x** does not occur **free** in the **function F**,

then **(λx.F x)** is **η-convertible** to **F**.

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

* CFG for the Lambda Calculus
* Function Abstraction
* Function Application
* Free and Bound Variables
* Beta Reductions
* Evaluating a Lambda Expression
* Currying
* Renaming Bound Variables by Alpha Reduction
* Eta Conversion
* Substitutions
* **Disambiguating Lambda Expressions**
* Normal Form
* Evaluation Strategies

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

Disambiguating Lambda Expressions

The grammar we gave in **function application** section

for lambda expressions is **ambiguous**.

A few simple rules will remove the **ambiguities**.

Function application is left associative: **f g h = ((f g) h)**

Function application binds more tightly than **lambda**:

λx.f g x = (λx.(f g) x)

The **body** in a **function abstraction** extends

as far to the right as possible: **λx. + x 1 = λx. (+ x 1).**

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

- CFG for the Lambda Calculus
- Function Abstraction
- Function Application
- Free and Bound Variables
- Beta Reductions
- Evaluating a Lambda Expression
- Currying
- Renaming Bound Variables by Alpha Reduction
- Eta Conversion
- Substitutions
- Disambiguating Lambda Expressions
- **Normal Form**
- Evaluation Strategies

http://www.cs.columbia.edu/~aho/cs4115/Lectures/15-04-13.html

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf

[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf