Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC
http://courses.engr.illinois.edu/cs421

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

α Conversion

- α-conversion:
  λ x. exp --α--> λ y. (exp [y/x])
- Provided that
  1. y is not free in exp
  2. No free occurrence of x in exp becomes bound in exp when replaced by y

α Conversion Non-Examples

1. Error: y is not free in termsecond

   λ x. x y ~~--α-->~~ λ y. y y

2. Error: free occurrence of x becomes bound in wrong way when replaced by y

   $\lambda x. \underbrace{\lambda y.\ x\ y}_{\text{exp}}\ \cancel{\text{--}\alpha\text{-->}}\ \lambda y. \underbrace{\lambda y.\ y\ y}_{\text{exp[y/x]}}$

But λ x. (λ y. y) x --α--> λ y. (λ y. y) y
And λ y. (λ y. y) y --α--> λ x. (λ y. y) x

Congruence

- Let ~ be a relation on lambda terms. ~ is a congruence if
- it is an equivalence relation
- If $e_1 \sim e_2$ then
  - $(e\ e_1) \sim (e\ e_2)$ and $(e_1 e) \sim (e_2\ e)$
  - $\lambda x.\ e_1 \sim \lambda x.\ e_2$

α Equivalence

- α equivalence is the smallest congruence containing α conversion
- One usually treats α-equivalent terms as equal - i.e. use α equivalence classes of terms

Example

Show: λ x. (λ y. y x) x ~α~ λ y. (λ x. x y) y

- λ x. (λ y. y x) x --α--> λ z. (λ y. y z) z so
  λ x. (λ y. y x) x ~α~ λ z. (λ y. y z) z
- (λ y. y z) --α--> (λ x. x z) so
  (λ y. y z) ~α~ (λ x. x z) so
  λ z. (λ y. y z) z ~α~ λ z. (λ x. x z) z
- λ z. (λ x. x z) z --α--> λ y. (λ x. x y) y so
  λ z. (λ x. x z) z ~α~ λ y. (λ x. x y) y
- λ x. (λ y. y x) x ~α~ λ y. (λ x. x y) y

Substitution

- Defined on α-equivalence classes of terms
- P [N / x] means replace every free occurrence of x in P by N
  - P called *redex*; N called *residue*
- Provided that no variable free in P becomes bound in P [N / x]
  - Rename bound variables in P to avoid capturing free variables of N

Substitution

- x [N / x] = N
- y [N / x] = y if y $\neq$ x
- (e_1 e_2) [N / x] = ((e_1 [N / x]) (e_2 [N / x]))
- (λ x. e) [N / x] = (λ x. e)
- (λ y. e) [N / x] = λ y. (e [N / x]) provided y $\neq$ x and y not free in N
  - Rename y in redex if necessary

Example

(λ y. y z) [(λ x. x y) / z] = ?

- Problems?
  - z in redex in scope of y binding
  - y free in the residue
- (λ y. y z) [(λ x. x y) / z] --α--> (λ w.w z) [(λ x. x y) / z] = λ w. w (λ x. x y)

Example

- Only replace free occurrences
- (λ y. y z (λ z. z)) [(λ x. x) / z] = λ y. y (λ x. x) (λ z. z)

Not

λ y. y (λ x. x) (λ z. (λ x. x))

β reduction

- β Rule: (λ x. P) N --β--> P [N /x]
- Essence of computation in the lambda calculus
- Usually defined on α-equivalence classes of terms

Example

- (λ z. (λ x. x y) z) (λ y. y z)
  --β--> (λ x. x y) (λ y. y z)
  --β--> (λ y. y z) y --β--> y z
- (λ x. x x) (λ x. x x)
  --β--> (λ x. x x) (λ x. x x)
  --β--> (λ x. x x) (λ x. x x) --β--> ....

α β Equivalence

- α β equivalence is the smallest congruence containing α equivalence and β reduction
- A term is in *normal form* if no subterm is α equivalent to a term that can be β reduced
- Hard fact (Church-Rosser): if e_1 and e_2 are $\alpha\beta$-equivalent and both are normal forms, then they are α equivalent

Order of Evaluation

- Not all terms reduce to normal forms
- Not all reduction strategies will produce a normal form if one exists

Lazy evaluation:

- Always reduce the left-most application in a top-most series of applications (i.e. Do not perform reduction inside an abstraction)
- Stop when term is not an application, or left-most application is not an application of an abstraction to a term

Example 1

- $(\lambda\ z.\ (\lambda\ x.\ x))\ ((\lambda\ y.\ y\ y)\ (\lambda\ y.\ y\ y))$
- Lazy evaluation:
- Reduce the left-most application:
- $(\lambda\ z.\ (\lambda\ x.\ x))\ ((\lambda\ y.\ y\ y)\ (\lambda\ y.\ y\ y))$
  --β--> $(\lambda\ x.\ x)$

Eager evaluation

- (Eagerly) reduce left of top application to an abstraction
- Then (eagerly) reduce argument
- Then β-reduce the application

Example 1

- $(\lambda\ z.\ (\lambda\ x.\ x))((\lambda\ y.\ y\ y)\ (\lambda\ y.\ y\ y))$
- Eager evaluation:
- Reduce the rator of the top-most application to an abstraction: Done.
- Reduce the argument:
- $(\lambda\ z.\ (\lambda\ x.\ x))((\lambda\ y.\ y\ y)\ (\lambda\ y.\ y\ y))$

--β--> $(\lambda\ z.\ (\lambda\ x.\ x))((\lambda\ y.\ y\ y)\ (\lambda\ y.\ y\ y))$

--β--> $(\lambda\ z.\ (\lambda\ x.\ x))((\lambda\ y.\ y\ y)\ (\lambda\ y.\ y\ y))$…

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. x x)((λ y. y y) (λ z. z)) --β-->

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. $\boxed{x}$ $\boxed{x}$)((λ y. y y) (λ z. z)) --β-->

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. $\boxed{x}$ $\boxed{x}$)((λ y. y y) (λ z. z)) --β-->

$\boxed{\text{((λ y. y y) (λ z. z))}}$ $\boxed{\text{((λ y. y y) (λ z. z))}}$

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. x x)((λ y. y y) (λ z. z)) --β-->

$\boxed{\text{((λ y. y y) (λ z. z))}}$ ((λ y. y y) (λ z. z)

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. x x)((λ y. y y) (λ z. z)) --β-->

((λ y. $\boxed{y}$ $\boxed{y}$) (λ z. z)) ((λ y. y y) (λ z. z))

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. x x)((λ y. y y) (λ z. z)) --β-->

((λ y. $\boxed{y}$ $\boxed{y}$) (λ z. z)) ((λ y. y y) (λ z. z))

--β--> ($\boxed{\text{(λ z. z)}}$ $\boxed{\text{(λ z. z)}}$)((λ y. y y) (λ z. z))

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. x x)((λ y. y y) (λ z. z)) --β-->
((λ y. y y) (λ z. z)) ((λ y. y y) (λ z. z))
--β--> ((λ z. z) (λ z. z))((λ y. y y) (λ z. z))

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. x x)((λ y. y y) (λ z. z)) --β-->
((λ y. y y) (λ z. z)) ((λ y. y y) (λ z. z))
--β--> ((λ z. z) (λ z. z))((λ y. y y) (λ z. z))

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. x x)((λ y. y y) (λ z. z)) --β-->
((λ y. y y) (λ z. z)) ((λ y. y y) (λ z. z))
--β--> ((λ z. z) (λ z. z))((λ y. y y) (λ z. z))
--β--> (λ z. z) ((λ y. y y) (λ z. z))

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. x x)((λ y. y y) (λ z. z)) --β-->
((λ y. y y) (λ z. z)) ((λ y. y y) (λ z. z))
--β--> ((λ z. z) (λ z. z))((λ y. y y) (λ z. z))
--β--> (λ z. z) ((λ y. y y) (λ z. z)) --β-->
(λ y. y y) (λ z. z)

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. x x)((λ y. y y) (λ z. z)) --β-->
((λ y. y y) (λ z. z)) ((λ y. y y) (λ z. z))
--β--> ((λ z. z) (λ z. z))((λ y. y y) (λ z. z))
--β--> (λ z. z) ((λ y. y y) (λ z. z)) --β-->
(λ y. y y) (λ z. z)

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Lazy evaluation:

(λ x. x x)((λ y. y y) (λ z. z)) --β-->
((λ y. y y) (λ z. z)) ((λ y. y y) (λ z. z))
--β--> ((λ z. z) (λ z. z))((λ y. y y) (λ z. z))
--β--> (λ z. z) ((λ y. y y) (λ z. z)) --β-->
(λ y. y y) (λ z. z) ~β~ λ z. z

Example 2

- (λ x. x x)((λ y. y y) (λ z. z))
- Eager evaluation:

(λ x. x x) $\boxed{\text{((λ y. y y) (λ z. z))}}$ --β-->
(λ x. x x) $\boxed{\text{((λ z. z) (λ z. z))}}$ --β-->
(λ x. x x) $\boxed{\text{(λ z. z)}}$ --β-->
(λ z. z) (λ z. z) --β--> λ z. z

η (Eta) Reduction

- η Rule: λ x. f x --η--> f if x not free in f
  - Can be useful in each direction
  - Not valid in Ocaml
    - recall lambda-lifting and side effects
  - Not equivalent to (λ x. f) x --> f (inst of β)
- Example: λ x. (λ y. y) x --η--> λ y. y