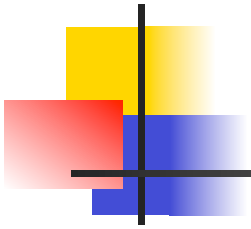
Programming Languages and Compilers (CS 421)

Elsa L Gunter

2112 SC, UIUC

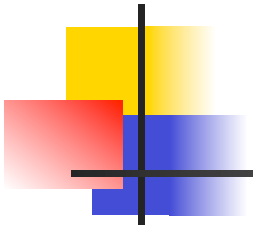
http://courses.engr.illinois.edu/cs421

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



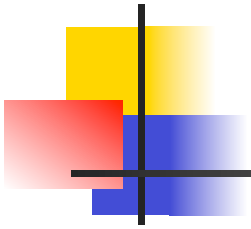
Lambda Calculus - Motivation

- Aim is to capture the essence of functions, function applications, and evaluation
- λ–calculus is a theory of computation
- "The Lambda Calculus: Its Syntax and Semantics". H. P. Barendregt. North Holland, 1984



Lambda Calculus - Motivation

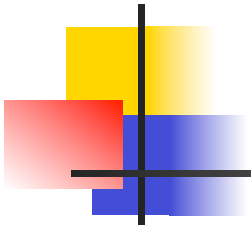
- All *sequential programs* may be viewed as functions from input (initial state and input values) to output (resulting state and output values).
- λ-calculus is a mathematical formalism of functions and functional computations
- Two flavors: typed and untyped



Untyped λ-Calculus

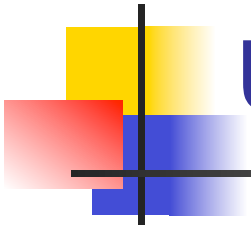
- Only three kinds of expressions:
  - Variables: x, y, z, w, …
  - Abstraction: λ x. e

    (Function creation, think fun x -> e)
  - Application: e_1 e_2



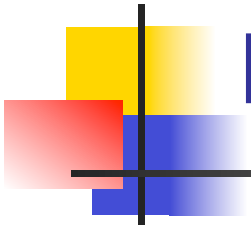
Untyped λ-Calculus Grammar

- Formal BNF Grammar:
  - <expression> ::= <variable>
    | <abstraction>
    | <application>
    | (<expression>)
  - <abstraction>
    ::= λ<variable>.<expression>
  - <application>
    ::= <expression> <expression>



Untyped λ-Calculus Terminology

- Occurrence: a location of a subterm in a term
- Variable binding: λ x. e is a binding of x in e
- Bound occurrence: all occurrences of x in λ x. e
- Free occurrence: one that is not bound
- Scope of binding: in λ x. e, all occurrences in e not in a subterm of the form λ x. e' (same x)
- Free variables: all variables having free occurrences in a term



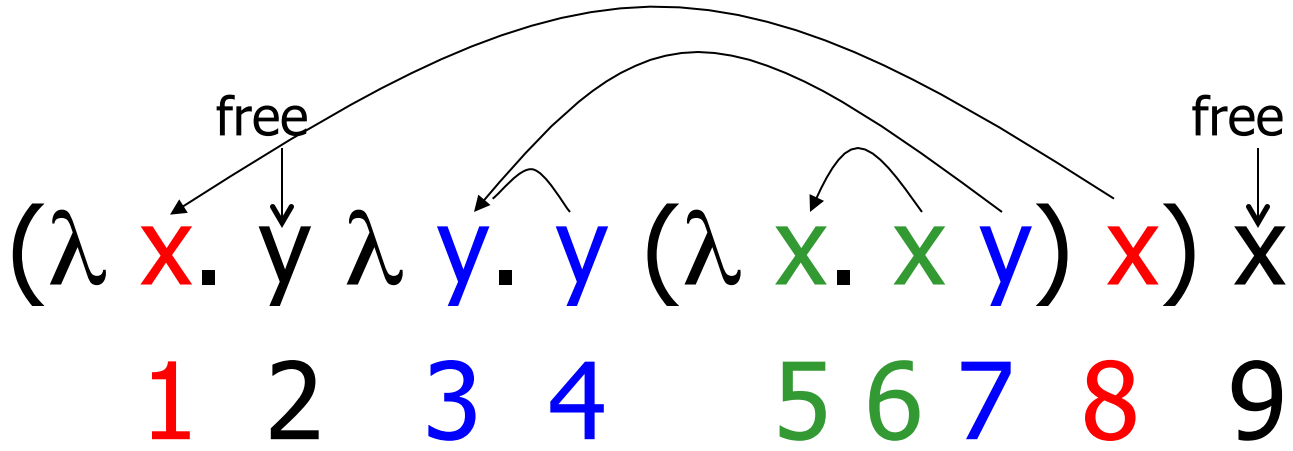
Example

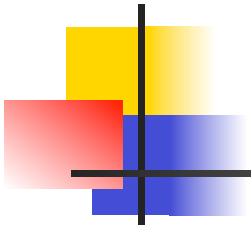
- Label occurrences and scope:

$$(\lambda\ x.\ y\ \lambda\ y.\ y\ (\lambda\ x.\ x\ y)\ x)\ x$$
$$1\ \ 2\ \ \ 3\ \ 4\ \ \ \ \ 5\ 6\ 7\ \ 8\ \ \ 9$$

Example

- Label occurrences and scope:





Untyped λ-Calculus

- How do you compute with the λ-calculus?
- Roughly speaking, by substitution:

- $(\lambda\ x.\ e_1)\ e_2 \Rightarrow^* e_1\,[e_2 / x]$

- * Modulo all kinds of subtleties to avoid free variable capture

Transition Semantics for λ-Calculus

$$\frac{E \text{ --> } E''}{E\ E' \text{ --> } E''\ E'}$$

- Application (version 1 - Lazy Evaluation)

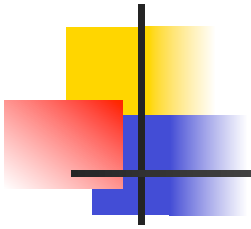
$$(\lambda\ x\ .\ E)\ E' \text{ --> } E[E'/x]$$

- Application (version 2 - Eager Evaluation)

$$\frac{E' \text{ --> } E''}{(\lambda\ x\ .\ E)\ E' \text{ --> } (\lambda\ x\ .\ E)\ E''}$$

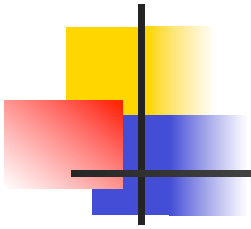
$$\frac{}{(\lambda\ x\ .\ E)\ V \text{ --> } E[V/x]}$$

V - variable or abstraction (value)



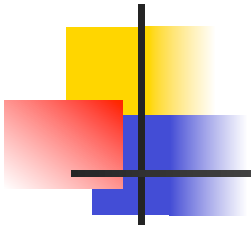
How Powerful is the Untyped λ-Calculus?

- The untyped λ-calculus is Turing Complete
  - Can express any sequential computation
- Problems:
  - How to express basic data: booleans, integers, etc?
  - How to express recursion?
  - Constants, if_then_else, etc, are conveniences; can be added as syntactic sugar



Typed vs Untyped λ-Calculus

- The *pure* λ-calculus has no notion of type: (f f) is a legal expression
- Types restrict which applications are valid
- Types are not syntactic sugar! They disallow some terms
- Simply typed λ-calculus is less powerful than the untyped λ-Calculus: NOT Turing Complete (no recursion)

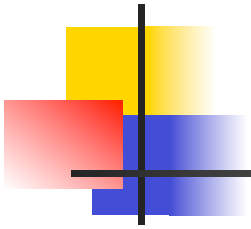


Uses of λ-Calculus

- Typed and untyped λ-calculus used for theoretical study of sequential programming languages
- Sequential programming languages are essentially the λ-calculus, extended with predefined constructs, constants, types, and syntactic sugar
- Ocaml is close to the λ-Calculus:

fun x -> exp --> λ x. exp

let x = e_1 in e_2 --> (λ x. e_2)e_1

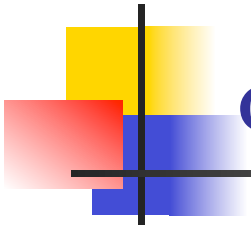


α Conversion

- 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

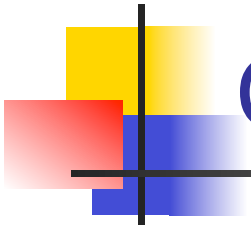
$$\lambda\ x.\ x\ y \text{ --}\alpha\text{--}\not>\ \lambda\ y.\ y\ y$$

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

$$\lambda\ x.\ \underbrace{\lambda\ y.\ x\ y}_{\text{exp}} \text{ --}\alpha\text{--}\not>\ \lambda\ y.\ \underbrace{\lambda\ y.\ y\ y}_{\text{exp[y/x]}}$$

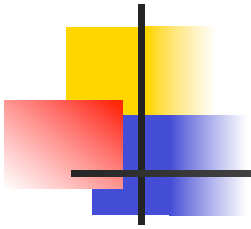
But $\lambda\ x.\ (\lambda\ y.\ y)\ x$ --α--> $\lambda\ y.\ (\lambda\ y.\ y)\ y$

And $\lambda\ y.\ (\lambda\ y.\ y)\ y$ --α--> $\lambda\ x.\ (\lambda\ y.\ y)\ x$



Congruence

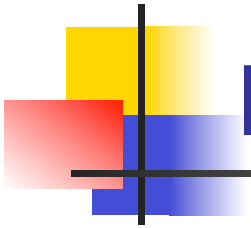
- Let ~ be a relation on lambda terms. ~ is a congruence if
- it is an equivalence relation
- If $e_1 \sim e_2$ then
  - $(e\ e_1) \sim (e\ e_2)$ and $(e_1 e) \sim (e_2\ e)$
  - $\lambda\ x.\ e_1 \sim \lambda\ x.\ e_2$



α Equivalence

- α equivalence is the smallest congruence containing α conversion

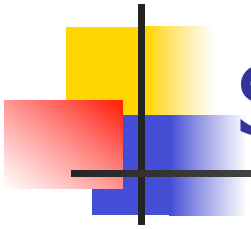
- One usually treats α-equivalent terms as equal - i.e. use α equivalence classes of terms



Example

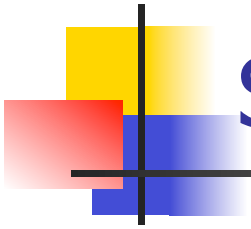
Show: λ x. (λ y. y x) x ~α~ λ y. (λ x. x y) y

- 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



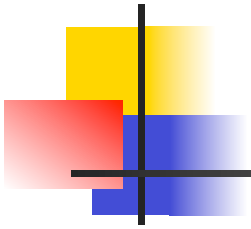
Substitution

- 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

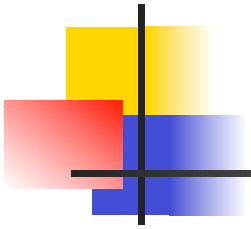
- $x\ [N / x] = N$
- $y\ [N / x] = y$ if $y \neq x$
- $(e_1\ e_2)\ [N / x] = ((e_1\ [N / x]\ )\ (e_2\ [N / x]\ ))$
- $(\lambda\ x.\ e)\ [N / x] = (\lambda\ x.\ e)$
- $(\lambda\ y.\ e)\ [N / x] = \lambda\ y.\ (e\ [N / x]\ )$
  provided $y \neq x$ and y not free in N
  - Rename y in redex if necessary



Example

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

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



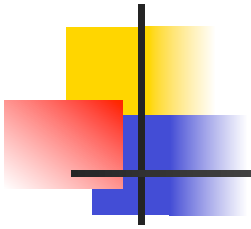
Example

- Only replace free occurrences
- $(\lambda\ y.\ y\ z\ (\lambda\ z.\ z))\ [(\lambda\ x.\ x)\ /\ z] =$

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

Not

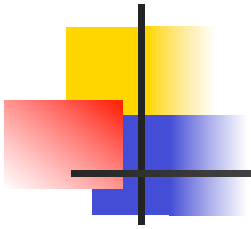
$$\lambda\ y.\ y\ (\lambda\ x.\ x)\ (\lambda\ z.\ (\lambda\ x.\ x))$$



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)

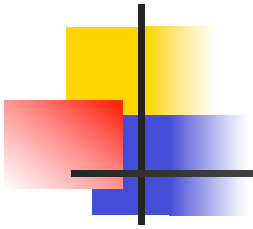
  --β--> (λ x. x y) (λ y. y z)

  --β--> (λ y. y z) y --β--> y z

- (λ x. x x) (λ x. x x)

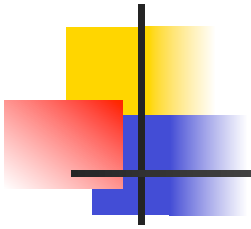
  --β--> (λ x. x x) (λ x. x x)

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



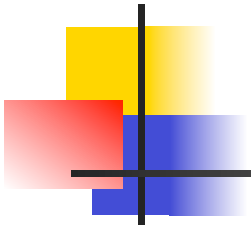
α β Equivalence

- 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

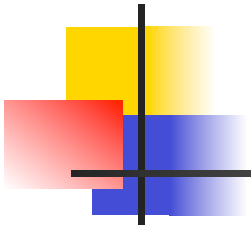
- Not all terms reduce to normal forms
- Not all reduction strategies will produce a normal form if one exists



Lazy evaluation:

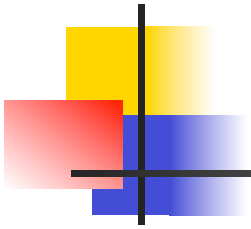
- Always reduce the left-most application in a top-most series of applications (i.e. Do not perform reduction inside an abstraction)

- Stop when term is not an application, or left-most application is not an application of an abstraction to a term



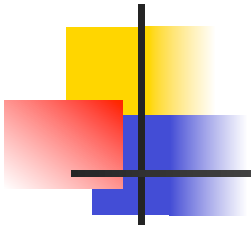
Example 1

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



Eager evaluation

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

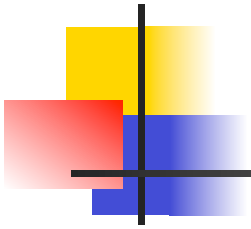


Example 1

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

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

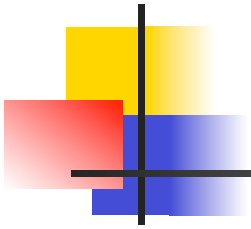
--β--> (λ z. (λ x. x))((λ y. y y) (λ y. y y))…



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:

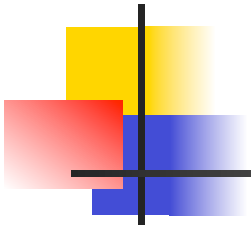
(λ x. x x)((λ y. y y) (λ z. z)) --β-->

Example 2

- $(\lambda\ x.\ x\ x)((\lambda\ y.\ y\ y)\ (\lambda\ z.\ z))$
- Lazy evaluation:

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

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

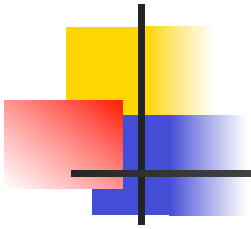


Example 2

- (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)



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))

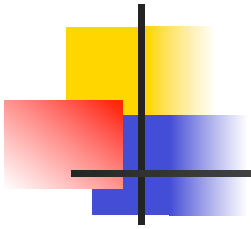
Example 2

- (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))



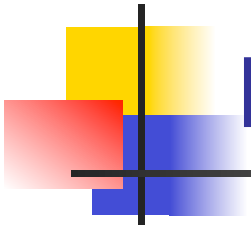
Example 2

- $(\lambda\ x.\ x\ x)((\lambda\ y.\ y\ y)\ (\lambda\ z.\ z))$
- Lazy evaluation:

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

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

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



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

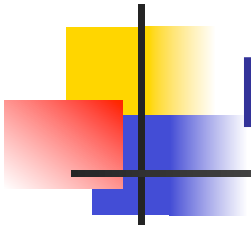
- $(\lambda\ x.\ x\ x)((\lambda\ y.\ y\ y)\ (\lambda\ z.\ z))$
- Lazy evaluation:

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

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

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

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



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:

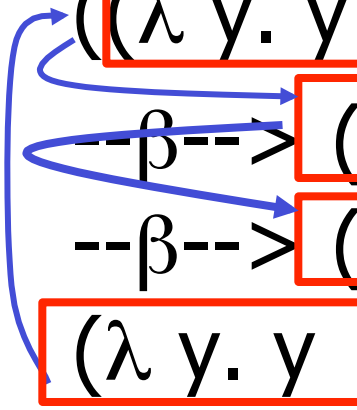
(λ x. x x)((λ y. y y) (λ z. z)) --β-->

((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) ((λ y. y y) (λ z. z)) --β-->

(λ x. x x) ((λ z. z) (λ z. z)) --β-->

(λ x. x x) (λ z. z) --β-->

(λ z. z) (λ z. z) --β--> λ z. z