

CS421 Lecture 8: λ -Calculus¹

Mark Hills
mhills@cs.uiuc.edu

University of Illinois at Urbana-Champaign

June 15, 2006

¹Based on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, and Elsa Gunter

Course Preview

Objectives

λ -Calculus

Computational Model

Data Representation

A Preview of Where We Are Going - 1 of 3

- ▶ Functional Programming
 - ▶ Intro to OCaml
 - ▶ Recursion
 - ▶ Higher-Order Functions
- ▶ Foundations of Programming Languages
 - ▶ Types and Type Systems
 - ▶ Lambda Calculus
 - ▶ Natural Semantics
 - ▶ Transition Semantics and the Church-Rosser Theorem

Preview - 2 of 3

- ▶ Language Syntax and Parsing
 - ▶ DFAs and NFAs
 - ▶ Grammars
 - ▶ First and Follow Sets
 - ▶ LL Grammars
 - ▶ LR Grammars
- ▶ Interpreters – Implementing Programming Language Semantics

Preview - 3 of 3

- ▶ Language Features for Organizing Data
 - ▶ Variables and Binding
 - ▶ Language Support for Data Abstraction
 - ▶ Object-Oriented Programming
- ▶ Language Features for Organizing Control
 - ▶ Logic Programming
 - ▶ Control Flow Basics: Structured, Unstructured, and Exceptions
 - ▶ Concurrency
 - ▶ Continuations and Continuation-Passing Style (CPS)
 - ▶ Function Calls and Parameter Passing

Objectives

Subtitle: "Everything is Really a Function"

This lecture introduces λ -calculus, and uses it to show how functions can represent arbitrary computations and data structures. After this lecture you should . . .

- ▶ Know the three constructs of λ -calculus.
- ▶ Know the terminology: free vs. bound variables, scopes, α -reduction, β -reduction
- ▶ Know how to use λ terms to represent arbitrary types.
 - ▶ numbers, booleans, pairs, etc

The λ-Calculus: Motivation

All *sequential* programs may be viewed as functions from input (initial state and input values) to output (resulting state and output values).

- ▶ The λ-calculus is a mathematical formalism of functions and functional computations
- ▶ Provides a simple, flexible model to represent programming languages
- ▶ Heavily used in languages research, and highly influential in functional programming

The λ-Calculus

λ-calculus : A language with only three kinds of expressions (called *terms*):

Variables: $e \rightarrow x$

Abstraction: $e \rightarrow \lambda x.e_1$ *i.e., Function creation*

Application: $e \rightarrow (e_1 e_2)$

Terminology

Occurrence : A location of a subterm in a term

Bound variable : Variable x is said to be bound in $\lambda x.e$.

Scope : The scope of this binding is all of e except any terms within e that are of the form $\lambda x.e_1$.

Bound occurrence : An occurrence of x in e is a *bound occurrence* if it is within the scope of some binding for x .

Free occurrence : Otherwise, an occurrence is said to be a *free occurrence* (aka *free variable*)

How powerful is the λ-Calculus?

Computing Power: .

- ▶ The language is Turing Complete – we can express any sequential computation
- ▶ “We don’t need no stinking integers!” – how do we represent basic data?
- ▶ How do we represent recursion?
- ▶ Constants, if-expressions, etc. are conveniences to make programming easier. *a.k.a., “syntactic sugar”*

How powerful is the λ-Calculus?

Typed vs. Untyped Lambda Calculus .

- ▶ Lambda Calculus has no notion of type. E.g., (ff) is a legal expression for any f !
- ▶ Types restrict what functions can be applied to what arguments
- ▶ Types are *not* just syntactic sugar: some terms are illegal in the typed calculus
- ▶ The simply typed λ-calculus is less powerful than the untyped λ-calculus – it is NOT Turing complete, since it lacks recursion

λ-Calculus in the Real World

- ▶ The typed and untyped λ-Calculi enable the theoretical study of sequential programming languages.
- ▶ A functional language is essentially the λ-Calculus extended with predefined constants, syntactic constructs, and types. E.g., Lisp, Scheme, functional subsets of ML and OCAML
- ▶ We can (mostly) express the λ-Calculus in OCaml:
 $\lambda x.x = \text{fun } x \rightarrow x$

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
--	--

Computing with the λ-Calculus

How do we “execute” programs in the λ-calculus? We need some method to systematically process λ terms to get some kind of “result”. This requires *reduction*, or *evaluation*, methods. We also need some way to tell when terms are *equivalent*. Finally, we need some notion of terms that are “answers”, so we know when to stop computing.

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
--	---

Congruence

Let \sim be a relation on lambda terms. \sim is a **congruence** if

- ▶ it is an equivalence relation
- ▶ $e_1 \sim e_2$ implies that $(e e_1) \sim (e e_2)$ and $(e_1 e) \sim (e_2 e)$ and $\lambda x. e_1 \sim \lambda x. e_2$

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
--	---

α Conversion

α-conversion is defined as:

$$\lambda x. e \rightarrow^\alpha \lambda y. (e[y/x])$$

provided

- ▶ y is not free in e
- ▶ no free occurrence of x becomes bound when replaced by y

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
--	---

α Conversion Non-Examples

- ▶ y is not free in e

$$\lambda x. x y \rightarrow^\alpha \lambda y. y y$$

- ▶ No free occurrence of x becomes bound when replaced by y

$$\lambda x. \lambda y. x y \rightarrow^\alpha \lambda y. \lambda y. y y$$

But, $\lambda x. (\lambda y. y) x \rightarrow^\alpha \lambda y. (\lambda y. y) y$; and,
 $\lambda y. (\lambda y. y) y \rightarrow^\alpha \lambda x. (\lambda y. y) x$

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
--	---

α Equivalence

α equivalence is the smallest congruence containing α conversion.

Generally, α-equivalent terms are considered equal – an individual term is a representative of an equivalence class of terms.

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
--	---

Example

Show: $\lambda x. (\lambda y. y x) x \sim_\alpha \lambda y. (\lambda x. x y) y$.

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
Example	
<p>Show: $\lambda x.(\lambda y.y x) x \sim_{\alpha} \lambda y.(\lambda x.x y) y$.</p> <ul style="list-style-type: none"> ▶ $\lambda x.(\lambda y.y x) x \rightarrow^{\alpha} \lambda z.(\lambda y.y z) z$ so $\lambda x.(\lambda y.y x) x \sim_{\alpha} \lambda z.(\lambda y.y z) z$ 	
◀ ▶ ↺ 🔍	
Mark Hills	CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
Example	
<p>Show: $\lambda x.(\lambda y.y x) x \sim_{\alpha} \lambda y.(\lambda x.x y) y$.</p> <ul style="list-style-type: none"> ▶ $\lambda x.(\lambda y.y x) x \rightarrow^{\alpha} \lambda z.(\lambda y.y z) z$ so $\lambda x.(\lambda y.y x) x \sim_{\alpha} \lambda z.(\lambda y.y z) z$ ▶ $\lambda y.y z \rightarrow^{\alpha} \lambda x.x z$ so $\lambda z.(\lambda y.y z) z \sim_{\alpha} \lambda z.(\lambda x.x z) z$ 	
◀ ▶ ↺ 🔍	
Mark Hills	CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
Example	
<p>Show: $\lambda x.(\lambda y.y x) x \sim_{\alpha} \lambda y.(\lambda x.x y) y$.</p> <ul style="list-style-type: none"> ▶ $\lambda x.(\lambda y.y x) x \rightarrow^{\alpha} \lambda z.(\lambda y.y z) z$ so $\lambda x.(\lambda y.y x) x \sim_{\alpha} \lambda z.(\lambda y.y z) z$ ▶ $\lambda y.y z \rightarrow^{\alpha} \lambda x.x z$ so $\lambda z.(\lambda y.y z) z \sim_{\alpha} \lambda z.(\lambda x.x z) z$ ▶ $\lambda z.(\lambda x.x z) z \rightarrow^{\alpha} \lambda y.(\lambda x.x y) y$ so $\lambda z.(\lambda x.x z) z \sim_{\alpha} \lambda y.(\lambda x.x y) y$ 	
◀ ▶ ↺ 🔍	
Mark Hills	CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
Example	
<p>Show: $\lambda x.(\lambda y.y x) x \sim_{\alpha} \lambda y.(\lambda x.x y) y$.</p> <ul style="list-style-type: none"> ▶ $\lambda x.(\lambda y.y x) x \rightarrow^{\alpha} \lambda z.(\lambda y.y z) z$ so $\lambda x.(\lambda y.y x) x \sim_{\alpha} \lambda z.(\lambda y.y z) z$ ▶ $\lambda y.y z \rightarrow^{\alpha} \lambda x.x z$ so $\lambda z.(\lambda y.y z) z \sim_{\alpha} \lambda z.(\lambda x.x z) z$ ▶ $\lambda z.(\lambda x.x z) z \rightarrow^{\alpha} \lambda y.(\lambda x.x y) y$ so $\lambda z.(\lambda x.x z) z \sim_{\alpha} \lambda y.(\lambda x.x y) y$ ▶ Thus $\lambda x.(\lambda y.y x) x \sim_{\alpha} \lambda y.(\lambda x.x y) y$. 	
◀ ▶ ↺ 🔍	
Mark Hills	CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
η Reduction	
<p>The η rule states that $\lambda x.f x \rightarrow^{\eta} f$ if x is not free in f</p> <ul style="list-style-type: none"> ▶ can be useful in each direction ▶ not valid in OCaml ▶ not equivalent to $(\lambda x.f) x$ (this is function application) <p>Example: $\lambda x.(\lambda y.y) x \rightarrow^{\eta} \lambda y.y$.</p>	
◀ ▶ ↺ 🔍	
Mark Hills	CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
Substitution	
<p>Substitution is defined over α-equivalence classes of terms.</p> $[N/x] P$ <p>means replace every free occurrence of x in P by N.</p> <ul style="list-style-type: none"> ▶ Requirement: No free variable in P should become bound in $[N/x] P$ – hence definition over α-equivalence classes ▶ Renaming/alpha conversion used to avoid name capture ▶ Often stated as a “hygiene” requirement for substitution, so we can assume it during discussions, but still need to be aware of it 	
◀ ▶ ↺ 🔍	
Mark Hills	CS421 Lecture 8: λ-Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Substitution Rules

Substitution defined inductively over the structure of λ terms

- ▶ $[N/x]x = N$
- ▶ $[N/x]y = y$ if $y \neq x$
- ▶ $[N/x](e_1 e_2) = ([N/x]e_1) ([N/x]e_2)$
- ▶ $[N/x]\lambda x.e = \lambda x.e$
- ▶ $[N/x]\lambda y.e = \lambda y.([N/x]e)$
if $y \neq x$ and y not free in N (rename y if needed to ensure)

Mark Hills CS421 Lecture 8: λ -Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Substitution Example

Any problems?

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

Mark Hills CS421 Lecture 8: λ -Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Substitution Example

Any problems?

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

- ▶ z in redex (term being substituted into) in scope of y binding
- ▶ y is free in the residue (term being substituted)

Mark Hills CS421 Lecture 8: λ -Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Substitution Example

Any problems?

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

- ▶ z in redex (term being substituted into) in scope of y binding
- ▶ y is free in the residue (term being substituted)

$$[(\lambda x.x y)/z](\lambda y.y z) \xrightarrow{\alpha} [(\lambda x.x y)/z](\lambda w.w z) = \lambda w.w(\lambda x.x y)$$

Mark Hills CS421 Lecture 8: λ -Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Substitution Example 2

How do we do substitution here?

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

Mark Hills CS421 Lecture 8: λ -Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Substitution Example 2

How do we do substitution here?
Is this correct?

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

Mark Hills CS421 Lecture 8: λ -Calculus

Substitution Example 2

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

How do we do substitution here?
 Is this correct?

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

No, this is, since we only replace *free* occurrences.

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

β Reduction

β reduction is the essence of computation in the λ calculus.

$$\beta \text{ rule: } (\lambda x.P) N \rightarrow^\beta [N/x]P$$

- Usually defined on α-equivalence classes of terms

β Reduction Example

$$(\lambda z.(\lambda x.x y) z)(\lambda y.y z) \rightarrow^\beta$$

β Reduction Example

$$(\lambda z.(\lambda x.x y) z)(\lambda y.y z) \rightarrow^\beta$$

$$(\lambda x.x y) (\lambda y.y z) \rightarrow^\beta$$

β Reduction Example

$$(\lambda z.(\lambda x.x y) z)(\lambda y.y z) \rightarrow^\beta$$

$$(\lambda x.x y) (\lambda y.y z) \rightarrow^\beta$$

$$(\lambda y.y z) y \rightarrow^\beta$$

β Reduction Example

$$(\lambda z.(\lambda x.x y) z)(\lambda y.y z) \rightarrow^\beta$$

$$(\lambda x.x y) (\lambda y.y z) \rightarrow^\beta$$

$$(\lambda y.y z) y \rightarrow^\beta$$

$$y z$$

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	---

β Reduction Example 2

$$(\lambda x. x x)(\lambda x. x x) \rightarrow^{\beta}$$

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ -Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	---

β Reduction Example 2

$$(\lambda x. x x)(\lambda x. x x) \rightarrow^{\beta}$$

$$(\lambda x. x x)(\lambda x. x x) \rightarrow^{\beta}$$

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ -Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	---

β Reduction Example 2

$$(\lambda x. x x)(\lambda x. x x) \rightarrow^{\beta}$$

$$(\lambda x. x x)(\lambda x. x x) \rightarrow^{\beta}$$

$$(\lambda x. x x)(\lambda x. x x) \rightarrow^{\beta}$$

$$(\lambda x. x x)(\lambda x. x x) \rightarrow^{\beta}$$

$$(\lambda x. x x)(\lambda x. x x) \rightarrow^{\beta}$$

$$(\lambda x. x x)(\lambda x. x x) \rightarrow^{\beta}$$

$$\dots$$

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ -Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	---

α β Equivalence

- ▶ α β equivalence is the smallest congruence containing α equivalence and β reduction
- ▶ **Definition:** a term is in *normal form* if no subterm is α equivalent to a term that can be β reduced – i.e. there is no way to reduce the term further
- ▶ hard fact (Church-Rosser): if e_1 and e_2 are $\alpha\beta$ -equivalent and both are normal forms, then they are α equivalent

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ -Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	---

Order of Evaluation

Order of Evaluation refers to the order of operations we choose to perform when reducing λ terms. For instance, should we substitute a λ term before or after reducing it? With this, there are two important points:

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

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ -Calculus

Outline Course Preview Objectives λ -Calculus Computational Model Data Representation	Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	---

Lazy Evaluation

- ▶ Always reduce the left-most, top-most application
- ▶ Stop when left-most term is not an application of an abstraction to a term

Note here that we choose to **not** evaluate under a λ – in some sense we require the term to be on “top”, so we don’t execute the bodies of functions before calling them.

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Eager Evaluation

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

Key point here: arguments reduced *before* β -reduction, not after

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Evaluation Order Example 1

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

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Evaluation Order Example 1

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

For lazy evaluation, first reduce the left-most application:

$$(\lambda z.(\lambda x.x))((\lambda y.y y)(\lambda y.y y)) \rightarrow^{\beta} (\lambda x.x)$$

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Evaluation Order Example 1

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

For eager evaluation, first reduce the left-most operator of the top-most application to an abstraction.

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

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Evaluation Order Example 1

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

For eager evaluation, first reduce the left-most operator of the top-most application to an abstraction.

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

It was, so no changes. Now, reduce the argument:

$$\begin{aligned} (\lambda z.(\lambda x.x))((\lambda y.y y)(\lambda y.y y)) &\rightarrow^{\beta} \\ (\lambda z.(\lambda x.x))((\lambda y.y y)(\lambda y.y y)) &\rightarrow^{\beta} \\ (\lambda z.(\lambda x.x))((\lambda y.y y)(\lambda y.y y)) &\rightarrow^{\beta} \\ &\dots \end{aligned}$$

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
---	--

Evaluation Order Example 2

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

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$(\lambda x. x x)((\lambda y. y y)(\lambda z. z z)) \rightarrow^{\beta}$$

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$(\lambda x. \boxed{x} \boxed{x})((\lambda y. y y)(\lambda z. z z)) \rightarrow^{\beta}$$

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$(\lambda x. \boxed{x} \boxed{x})((\lambda y. y y)(\lambda z. z z)) \rightarrow^{\beta}$$

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

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$(\lambda x. x x)((\lambda y. y y)(\lambda z. z z)) \rightarrow^{\beta}$$

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

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$(\lambda x. x x)((\lambda y. y y)(\lambda z. z z)) \rightarrow^{\beta}$$

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

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$(\lambda x. x x)((\lambda y. y y)(\lambda z. z z)) \rightarrow^{\beta}$$

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

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$\begin{aligned}
 & (\lambda x.x x)((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda y.y y)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & \boxed{(\lambda z.z)(\lambda z.z)} ((\lambda y.y y)(\lambda z.z))
 \end{aligned}$$

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$\begin{aligned}
 & (\lambda x.x x)((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda y.y y)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda z.\underline{z})(\lambda z.z)) ((\lambda y.y y)(\lambda z.z))
 \end{aligned}$$

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$\begin{aligned}
 & (\lambda x.x x)((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda y.y y)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda z.\underline{z})(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & \boxed{(\lambda z.z)} ((\lambda y.y y)(\lambda z.z))
 \end{aligned}$$

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$\begin{aligned}
 & (\lambda x.x x)((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda y.y y)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda z.z)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & (\lambda z.\underline{z}) ((\lambda y.y y)(\lambda z.z))
 \end{aligned}$$

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$\begin{aligned}
 & (\lambda x.x x)((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda y.y y)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda z.z)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & (\lambda z.\underline{z}) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & (\lambda y.y y)(\lambda z.z)
 \end{aligned}$$

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α β Equivalence Evaluation Order
--	--

Evaluation Order Example 2

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

Lazy evaluation:

$$\begin{aligned}
 & (\lambda x.x x)((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda y.y y)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda z.z)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & (\lambda z.\underline{z}) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & (\lambda y.y y)(\lambda z.z) \rightarrow^{\beta} \\
 & \lambda z.z
 \end{aligned}$$

◀ ▶ ↺ ↻ 🔍

Mark Hills CS421 Lecture 8: λ-Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
---	---

Evaluation Order Example 2

Lazy evaluation:

$$\begin{aligned}
 & (\lambda x.x x)((\lambda y.y y)(\lambda z.z)) \\
 & (\lambda x.x x)((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda y.y y)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & ((\lambda z.z)(\lambda z.z)) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & (\lambda z.\underline{z}) ((\lambda y.y y)(\lambda z.z)) \rightarrow^{\beta} \\
 & (\lambda y.y y)(\lambda z.z) \rightarrow^{\beta} \\
 & (\lambda z.z)(\lambda z.z) \rightarrow^{\beta} \\
 & \lambda z.z \sim_{\beta} ((\lambda y.y y)(\lambda z.z))
 \end{aligned}$$

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Terminology Congruence α Conversion α Equivalence η Reduction Substitution β Reduction α, β Equivalence Evaluation Order
---	---

Evaluation Order Example 2

Eager evaluation:

$$\begin{aligned}
 & (\lambda x.x x)((\lambda y.y y)(\lambda z.z)) \\
 & (\lambda x.x x) \boxed{((\lambda y.y y)(\lambda z.z))} \rightarrow^{\beta} \\
 & (\lambda x.x x) \boxed{((\lambda z.z)(\lambda z.z))} \rightarrow^{\beta} \\
 & (\lambda x.x x) \boxed{(\lambda z.z)} \rightarrow^{\beta} \\
 & (\lambda z.z)(\lambda z.z)
 \end{aligned}$$

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Where's the Data? Data Structures, Take 1 Data Structures, Take 2 Data Structures, Take 3 Church Numerals Primitive Recursion Recursion
---	---

But...

Great, we can compute! But with what? We have functions, but we don't have data yet!

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Where's the Data? Data Structures, Take 1 Data Structures, Take 2 Data Structures, Take 3 Church Numerals Primitive Recursion Recursion
---	---

But...

Great, we can compute! But with what? We have functions, but we don't have data yet!

- ▶ **Key Concept:** We can encode data as functions!

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Where's the Data? Data Structures, Take 1 Data Structures, Take 2 Data Structures, Take 3 Church Numerals Primitive Recursion Recursion
---	--

How to Represent (Free) Data Structures (First Pass)

- ▶ Suppose τ is a type with n constructors with no arguments

$$C_1, \dots, C_n$$
- ▶ We can represent each term as an abstraction

$$C_i \rightarrow \lambda x_1 \dots \lambda x_n. x_i$$
- ▶ Think: you give me what to return in each case (like a match statement), I'll apply the correct case based on how I was constructed

Mark Hills CS421 Lecture 8: λ -Calculus

<ul style="list-style-type: none"> Outline Course Preview Objectives λ-Calculus Computational Model Data Representation 	<ul style="list-style-type: none"> Where's the Data? Data Structures, Take 1 Data Structures, Take 2 Data Structures, Take 3 Church Numerals Primitive Recursion Recursion
---	--

How to Represent Booleans

For bool = True | False

- ▶ True $\rightarrow \lambda x.\lambda y.x$
- ▶ False $\rightarrow \lambda x.\lambda y.y$

Mark Hills CS421 Lecture 8: λ -Calculus

Some New Notation

From here out, we will write

- ▶ $\lambda x_1 x_2 \dots x_n. e$ for $\lambda x_1. \lambda x_2. \dots \lambda x_n. e$
- ▶ $e_1 e_2 \dots e_n$ for $((e_1 e_2) \dots e_n)$

How to Write Functions over Data Structures

We have basic data, but we need to be able to use them. We still need some kind of case or match functionality.

```
1 match c with
2 | C_1 -> x_1
3 | C_2 -> x_2
4 ...
5 | C_n -> x_n
```

We can encode this as

$$\lambda c x_1 x_2 \dots x_n. c x_1 x_2 \dots x_n$$

- ▶ **Key Idea:** give me what to do in each case and give me a case, and I'll apply that case

How to Write Functions over Booleans

How do we represent conditionals?

- ▶ if b then c else d
- ▶ Represent as

$$\text{if_then_else} \equiv \lambda b c d. b c d$$

Booleans Example

$$\begin{aligned} \text{not } b &= \text{if } b \text{ then False else True} \\ &= \text{if_then_else } b \text{ False True} \\ &= (\lambda b c d. b c d) b (\lambda x y.y)(\lambda x y.x) \\ &= b(\lambda x y.y)(\lambda x y.x) \end{aligned}$$

So, lift this to:

$$\text{not} \equiv \lambda b. b(\lambda x y.y)(\lambda x y.x)$$

And and Or are similar, you should try to devise them yourself...

How to Represent (Free) Data Structures (Second Pass)

- ▶ Suppose τ is a type with n constructors

$$C_1 t_{11} \dots t_{1k}, \dots, C_n t_{n1} \dots t_{nm}$$

- ▶ Represent each term as an abstraction:

$$C_i t_{i1} \dots t_{ij} \rightarrow \lambda x_1 \dots x_n. x_i t_{i1} \dots t_{ij}$$

or

$$C_i \rightarrow \lambda t_{i1} \dots t_{ij} x_1 \dots x_n. x_i t_{i1} \dots t_{ij}$$

- ▶ **Key Idea:** You need to give each constructor its arguments first

How to Represent Pairs

Pair has one constructor, the comma, that takes two arguments.

- ▶ $(a, b) \rightarrow \lambda x. x a b$
- ▶ $(_, _) \rightarrow \lambda a b x. x a b$

Functions over Pairs

► $\text{fst} \equiv \lambda p.p(\lambda x y.x)$

$$\begin{aligned} \text{fst}(u, v) &\rightarrow \\ (\lambda p.p(\lambda x y.x))((\lambda a b x.x a b) u v) &\rightarrow \\ (\lambda p.p(\lambda x y.x))(\lambda x.x u v) &\rightarrow \\ (\lambda x.x u v)(\lambda x y.x) &\rightarrow \\ (\lambda x y.x) u v &\rightarrow \\ (\lambda y.u) v &\rightarrow \\ u \end{aligned}$$

► $\text{snd} \equiv \lambda p.p(\lambda x y.y)$

How to Represent (Free) Data Structures (Third Pass)

► Suppose τ is a type with n constructors

$$C_1 t_{11} \cdots t_{1k}, \dots, C_n t_{n1} \cdots t_{nm}$$

► Suppose $t_{ih} : \tau$ (i.e. it is recursive); then, in place of a value t_{ih} we use a function to compute the recursive value $t_{ih} x_1 \cdots x_n$

$$C_i \rightarrow \lambda t_{i1} \cdots t_{ij} x_1 \cdots x_n. x_i t_{i1} \cdots (t_{ij} x_1 \cdots x_n) \cdots t_{ij}$$

Representing Natural Numbers

► $\text{nat} \equiv \text{Suc nat} \mid 0$

► $0 \equiv \lambda f x.x$

► $\text{Suc } n \equiv \lambda f x.f(n f x)$

► $\text{Suc} \equiv \lambda n f x.f(n f x)$

This numeric representation is referred to as *Church Numerals* for Alonzo Church, the creator of the λ -calculus.

Some Church Numerals

$$\begin{aligned} \text{Suc } 0 &= (\lambda n f x.f(n f x))(\lambda f x.x) \rightarrow \\ &\lambda f x.f((\lambda f x.x)f x) \rightarrow \\ &\lambda f x.f((\lambda x.x)x) \rightarrow \\ &\lambda f x.f x \end{aligned}$$

So, this says to apply the function to its argument once. This defines numbers, in some sense, in terms of their *potential* – what can we do with 0 of something, or 1 of something, 2, etc.

Some Church Numerals

$$\begin{aligned} \text{Suc}(\text{Suc } 0) &= (\lambda n f x.f(n f x))(\text{Suc } 0) \rightarrow \\ &(\lambda n f x.f(n f x))(\lambda f x.f x) \rightarrow \\ &\lambda f x.f((\lambda f x.f x)f x) \rightarrow \\ &\lambda f x.f((\lambda x.f x)x) \rightarrow \\ &\lambda f x.f(f x) \end{aligned}$$

In general, for any number n , we will apply function f a total of n times.

Adding Church Numerals

For two numbers n and m , we will have Church numerals $\lambda f x.f^n x$ and $\lambda f x.f^m x$ respectively. Adding the two, we would expect to get something of the form $\lambda f x.f^{n+m} x$. What would this look like?

$$\begin{aligned} n + m &= \lambda f x.f^{n+m} x = \\ &\lambda f x.f^n(f^m x) = \\ &\lambda f x.n f(m f x) \\ &\text{or} \\ + &\equiv \lambda n m f x.n f(m f x) \end{aligned}$$

Multiplying Church Numerals

For two numbers n and m , we will have Church numerals $\lambda f x.f^n x$ and $\lambda f x.f^m x$ respectively. Multiplying the two, we would expect to get something of the form $\lambda f x.f^{n \times m} x$. What would this look like?

$$\begin{aligned} n \times m &= \lambda f x.f^{n \times m} x = \\ & \lambda f x.(f^m)^n x = \\ & \lambda f x.n(m f)x \\ & \text{or} \\ & \times \equiv \lambda n m f x.n(m f)x \end{aligned}$$

Primitive Recursion over Nat

Recall folding:

```
1 fold f z n =
2   match n with 0 -> z
3               | Suc m -> f (fold f z m)
```

We can represent this as:

$$\text{fold} \equiv \lambda f z n.n f z$$

For instance:

```
is_zero n = fold(λr.False)True n =
(λf x.f^n x)(λr.False)True =
((λr.False)^n)True =
if n = 0 then True else False
```

Predecessor

Predecessor is, surprisingly, rather painful. First we define an auxiliary function that calculates the predecessor as a pair – the predecessor of 20 would, as a pair, be listed as (20, 19):

```
let pred_aux n =
  match n with 0 = (0,0) | Suc m =
  (Suc(fst(pred_aux m)), fst(pred_aux m)) =
  fold(λr.(Suc(fst r), fst r)) (0,0) n
```

With this, we can now devise a predecessor function directly:

```
pred ≡ λn.snd(pred_aux n)n =
λn.snd(fold(λr.(Suc(fst r), fst r)) (0,0) n)
```

Recursion

To support recursion *directly*, versus by just using textual substitution, we want a λ -term Y such that for any term R the following holds:

$$Y R = R(YR)$$

Y needs to use replication to “remember” R .

```
Y = λy.(λx.y(x x))(λx.y(x x))
Y R = (λx.R(x x))(λx.R(x x)) =
R((λx.R(x x))(λx.R(x x)))
```

Note that this **requires** lazy evaluation!

Factorial

We can define factorial as $F = \lambda f n.\text{if } n = 0 \text{ then } 1 \text{ else } n \times f(n-1)$. Now,

$$\begin{aligned} Y F 3 &= F(Y F) 3 = \\ & \text{if } 3 = 0 \text{ then } 1 \text{ else } 3 \times ((Y F)(3-1)) = \\ & 3 \times (Y F) 2 = \\ & 3 \times (F(Y F) 2) = \\ & 3 \times (\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 \times (Y F)(2-1)) = \\ & 3 \times (2 \times (Y F)1) = \\ & 3 \times (2 \times (F(Y F)1)) = \dots \\ & 3 \times (2 \times (1 \times (\text{if } 0 = 0 \text{ then } 1 \text{ else } 0 \times (Y F)(0-1)))) = \\ & 3 \times 2 \times 1 \times 1 = 6 \end{aligned}$$

Y in OCaml

```
1 # let rec y f = f (y f);;
2 val y : ('a -> 'a) -> 'a = <fun>
3 # let mk_fact =
4   fun f n -> if n = 0 then 1 else n * f (n - 1);;
5 val mk_fact : (int -> int) -> int -> int = <fun>
6 # y mk_fact;;
7 Stack overflow during evaluation (looping recursion?).
```

Eager Eval Y in OCaml

```
1 # let rec y f x = f (y f) x;;  
2 val y : (('a -> 'b) -> 'a -> 'b) -> 'a -> 'b = <fun>  
3 # y mk_fact;;  
4 - : int -> int = <fun>  
5 # y mk_fact 5;;  
6 - : int = 120
```

Some other Combinators

- ▶ $I = \lambda x.x$
- ▶ $K = \lambda x.\lambda y.x$
- ▶ $K_* = \lambda x.\lambda y.y$
- ▶ $S = \lambda x.\lambda y.\lambda z.x z (y z)$