

# Untyped Lambda Calculus

Structure and Evaluation, Currying, Church Encodings

---

Sven Tennie

July 29, 2018

Dream IT

<https://dreamit.de>

# Introduction

---

# Lambda Calculus

- Invented by Alonzo Church (1920s)
- Equally expressive to the Turing Machine(s)
- Formal Language
- Computational Model
  - Lisp (1950s)
  - ML
  - Haskell
- "Lambda Expressions" in almost every modern programming language

# Why should I care?

- Simple Computational Model
  - to describe structure and behaviour (E.g. Operational Semantics, Type Systems)
  - to reason and prove

# Why should I care?

- Simple Computational Model
  - to describe structure and behaviour (E.g. Operational Semantics, Type Systems)
  - to reason and prove
- Explains why things in FP are like they are
  - pure functions
  - higher-order functions
  - currying
  - lazy evaluation

# Why should I care?

- Simple Computational Model
  - to describe structure and behaviour (E.g. Operational Semantics, Type Systems)
  - to reason and prove
- Explains why things in FP are like they are
  - pure functions
  - higher-order functions
  - currying
  - lazy evaluation
- Understand FP Compilers
  - Introduce FP stuff into other languages
  - Write your own compiler
  - GHC uses an enriched Lambda Calculus internally

# Basics

---

# Untyped Lambda Calculus

$t ::=$

$x$

$\lambda x. t$

$t \ t$

Terms:

Variable

Abstraction

Application



# Untyped Lambda Calculus

$t ::=$

$x$

$\lambda x.t$

$t\ t$

Terms:

Variable

Abstraction

Application

## Example - Identity

### Lambda Calculus

$\underbrace{\underbrace{\lambda x.x}_{\text{Abstraction}} \underbrace{y}_{\text{Variable}}}_{\text{Application}} \rightarrow y$

# Untyped Lambda Calculus

$t ::=$

$x$

$\lambda x. t$

$t \ t$

Terms:

Variable

Abstraction

Application

## Example - Identity

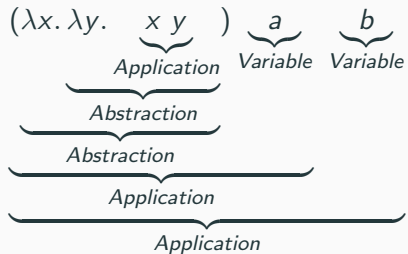
### Lambda Calculus

$\underbrace{\underbrace{\lambda x. x}_{\text{Abstraction}} \underbrace{y}_{\text{Variable}}}_{\text{Application}} \rightarrow y$

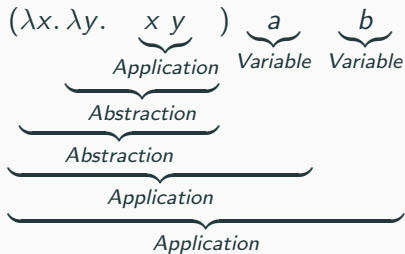
### Javascript

$\underbrace{(\underbrace{\text{function } (x) \{ \text{return } x; \}}_{\text{Abstraction}}) (\underbrace{y}_{\text{Variable}})}_{\text{Application}}$

# Evaluation / Reduction



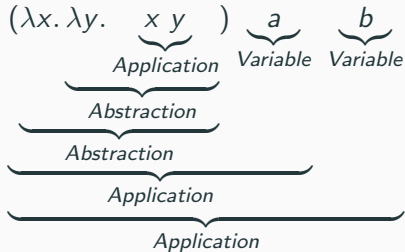
# Evaluation / Reduction



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

Parentheses are not part of the grammar? See next slide... :)

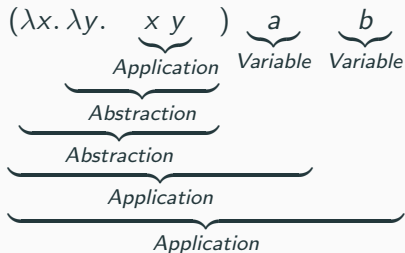
# Evaluation / Reduction



$$\begin{aligned} & (\lambda x. \lambda y. x y) a b \\ \rightarrow & (\lambda y. a y) b \end{aligned}$$

Parentheses are not part of the grammar? See next slide... :)

# Evaluation / Reduction


$$\begin{aligned} & (\lambda \textcolor{brown}{x}. \lambda y. \textcolor{brown}{x} \ y) \textcolor{brown}{a} \textcolor{brown}{b} \\ \rightarrow & (\lambda \textcolor{blue}{y}. \textcolor{blue}{a} \ \textcolor{blue}{y}) \textcolor{blue}{b} \\ \rightarrow & \textcolor{brown}{a} \ \textcolor{brown}{b} \end{aligned}$$

Parentheses are not part of the grammar? See next slide... :)

# Notational Conventions

- We use parentheses to clarify what's meant
- Applications associate to the left

$$s\ t\ u \equiv (s\ t)\ u$$

- Abstractions expand as much to the right as possible

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

$\lambda x. \lambda y. x \ y \ z$

## *Bound and free*

$\lambda y$   $y$  is *bound*,  $x$  and  $z$  are *free*

$\lambda x$   $x$  and  $y$  are *bound*,  $z$  is *free*

$\lambda x, \lambda y$  *binder*



$\lambda x. \lambda y. x \ y \ z$

## *Bound and free*

$\lambda y \ y$  is *bound*,  $x$  and  $z$  are *free*

$\lambda x \ x$  and  $y$  are *bound*,  $z$  is *free*

$\lambda x, \lambda y$  *binder*

A term with no free variables is "*closed*"

- A "*combinator*"
- $id \equiv \lambda x. x$
- $Y, S, K, I \dots$

# Higher Order Functions

- Functions that take or return functions
  - Are there "by definition"

$$\underbrace{\underbrace{\lambda x.x}_{\text{Abstraction}} \underbrace{\lambda y.y}_{\text{Abstraction}}}_{\text{Application}} \rightarrow \underbrace{\lambda y.y}_{\text{Abstraction}}$$

## Idea

- Take a funktion with  $n$  arguments
- Create a funktion that takes one argument and returns a function with  $n - 1$  arguments

# Currying

## Idea

- Take a funktion with  $n$  arguments
- Create a funktion that takes one argument and returns a function with  $n - 1$  arguments

## Example

- (+1) Section in Haskell
- $(\lambda x. \lambda y. + x y) 1 \rightarrow \lambda y. + 1 y$

# Currying

## Idea

- Take a funktion with  $n$  arguments
- Create a funktion that takes one argument and returns a function with  $n - 1$  arguments

## Example

- (+1) Section in Haskell
- $(\lambda x. \lambda y. + x y) 1 \rightarrow \lambda y. + 1 y$
- Partial Function Application is there "by definition"
  - You can use this stunt to "curry" in every language that supports "Lambda Expressions"

## Alpha conversion

$$\lambda x.x \rightarrow_{\alpha} \lambda y.y$$

# Reductions and Conversions

## Alpha conversion

$$\lambda x.x \rightarrow_{\alpha} \lambda y.y$$

## Beta reduction

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

# Reductions and Conversions

## Alpha conversion

$$\lambda x.x \rightarrow_{\alpha} \lambda y.y$$

## Beta reduction

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

## Eta conversion

Iff (if and only if)  $x$  is not free in  $f$ :

$$\begin{aligned} \lambda x.f \ x &\rightarrow_{\eta} f \\ (\lambda x. \underbrace{(\lambda y.y)}_f \ x) \ a &\rightarrow_{\eta} \underbrace{(\lambda y.y)}_f \ a \end{aligned}$$

If  $x$  is free in  $f$ , no  $\eta$  conversion possible:

$$\lambda x. \left( \overset{\text{Bound}}{\downarrow} \lambda y.y \ \overset{\downarrow}{x} \right) \ x \not\rightarrow_{\eta} \left( \lambda y.y \ \overset{\text{Free?!}}{\downarrow} \overset{\downarrow}{x} \right)$$



- Everything (Term) is an Expression
  - No statements
- No "destructive" Assignments
  - The reason why FP Languages promote pure functions
  - But you could invent a built-in function to manipulate "state"...

# Evaluation

---

- We learned how to write down and talk about Lambda Calculus Terms
- How to evaluate them?
- Different Strategies
  - Interesting outcomes

# Full Beta-Reduction

- RedEx
  - **R**educible **E**xpression
  - Always an Application

$$\begin{array}{c} (\lambda x.x) \ ((\lambda x.x) \ (\lambda z. \underbrace{(\lambda x.x) \ z}_{\text{RedEx}})) \\ \underbrace{\hspace{10em}}_{\text{RedEx}} \end{array}$$

# Full Beta-Reduction

- RedEx
  - **R**educible **E**xpression
  - Always an Application

$$\underbrace{(\lambda x.x) \underbrace{((\lambda x.x) \underbrace{(\lambda z. (\lambda x.x) z))}_{RedEx})}_{RedEx}}_{RedEx}$$

## Full Beta-Reduction

- Any RedEx, Any Time
- Like in Arithmetics
- Too vague for programming. . .

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

## Normal Order Reduction

- Left-most, Outer-most RedEx

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

## Normal Order Reduction

- Left-most, Outer-most RedEx

# Normal Order Reduction

$$\begin{aligned} & (\lambda x.x) ((\lambda x.x) (\lambda z.(\lambda x.x) z)) \\ \rightarrow & (\lambda x.x) (\lambda z.(\lambda x.x) z) \end{aligned}$$

## Normal Order Reduction

- Left-most, Outer-most RedEx



# Normal Order Reduction

$$\begin{aligned} & (\lambda x.x) ((\lambda x.x) (\lambda z.(\lambda x.x) z)) \\ \rightarrow & (\lambda x.x) (\lambda z.(\lambda x.x) z) \end{aligned}$$

## Normal Order Reduction

- Left-most, Outer-most RedEx

# Normal Order Reduction

$$\begin{aligned} & (\lambda x.x) ((\lambda x.x) (\lambda z.(\lambda x.x) z)) \\ \rightarrow & (\lambda x.x) (\lambda z.(\lambda x.x) z) \\ \rightarrow & \lambda z.(\lambda x.x) z \end{aligned}$$

## Normal Order Reduction

- Left-most, Outer-most RedEx

# Normal Order Reduction

$$\begin{aligned} & (\lambda x.x) ((\lambda x.x) (\lambda z.(\lambda x.x) z)) \\ \rightarrow & (\lambda x.x) (\lambda z.(\lambda x.x) z) \\ \rightarrow & \lambda z.(\lambda x.x) z \end{aligned}$$

## Normal Order Reduction

- Left-most, Outer-most RedEx

# Normal Order Reduction

$$\begin{aligned} & (\lambda x.x) ((\lambda x.x) (\lambda z.(\lambda x.x) z)) \\ \rightarrow & (\lambda x.x) (\lambda z.(\lambda x.x) z) \\ \rightarrow & \lambda z.(\lambda x.x) z \\ \rightarrow & \lambda z.z \end{aligned}$$

## Normal Order Reduction

- Left-most, Outer-most RedEx

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

## Call-by-Name

- like Normal Order Reduction, but NO reductions inside Abstractions
  - Abstractions are values
- lazy, non-strict
- Parameters are NOT evaluated before they are used
- Optimization: Save result  $\rightarrow$  *Call-by-Need*

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

## Call-by-Name

- like Normal Order Reduction, but NO reductions inside Abstractions
  - Abstractions are values
- lazy, non-strict
- Parameters are NOT evaluated before they are used
- Optimization: Save result  $\rightarrow$  *Call-by-Need*

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

## Call-by-Name

- like Normal Order Reduction, but NO reductions inside Abstractions
  - Abstractions are values
- lazy, non-strict
- Parameters are NOT evaluated before they are used
- Optimization: Save result  $\rightarrow$  *Call-by-Need*

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

## Call-by-Name

- like Normal Order Reduction, but NO reductions inside Abstractions
  - Abstractions are values
- lazy, non-strict
- Parameters are NOT evaluated before they are used
- Optimization: Save result  $\rightarrow$  *Call-by-Need*



$$\begin{aligned} & (\lambda x.x) ((\lambda x.x) (\lambda z.(\lambda x.x) z)) \\ \rightarrow & (\lambda x.x) (\lambda z.(\lambda x.x) z) \\ \rightarrow & \lambda z.(\lambda x.x) z \end{aligned}$$

## Call-by-Name

- like Normal Order Reduction, but NO reductions inside Abstractions
  - Abstractions are values
- lazy, non-strict
- Parameters are NOT evaluated before they are used
- Optimization: Save result  $\rightarrow$  *Call-by-Need*

# Call-by-Name

$$\begin{aligned} & (\lambda x.x) ((\lambda x.x) (\lambda z.(\lambda x.x) z)) \\ \rightarrow & (\lambda x.x) (\lambda z.(\lambda x.x) z) \\ \rightarrow & \lambda z.(\lambda x.x) z \\ \not\rightarrow & \end{aligned}$$

## Call-by-Name

- like Normal Order Reduction, but NO reductions inside Abstractions
  - Abstractions are values
- lazy, non-strict
- Parameters are NOT evaluated before they are used
- Optimization: Save result  $\rightarrow$  *Call-by-Need*

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

## Call-by-Value

- Outer-most, only if right-hand side was reduced to a value
  - Parameters ARE evaluated before they are used
- NO reductions inside Abstractions
  - Abstractions are values
- eager, strict

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

## Call-by-Value

- Outer-most, only if right-hand side was reduced to a value
  - Parameters ARE evaluated before they are used
- NO reductions inside Abstractions
  - Abstractions are values
- eager, strict

$$\begin{aligned} & (\lambda x.x) ((\lambda x.x) (\lambda z.(\lambda x.x) z)) \\ \rightarrow & (\lambda x.x) (\lambda z.(\lambda x.x) z) \end{aligned}$$

## Call-by-Value

- Outer-most, only if right-hand side was reduced to a value
  - Parameters ARE evaluated before they are used
- NO reductions inside Abstractions
  - Abstractions are values
- eager, strict

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

## Call-by-Value

- Outer-most, only if right-hand side was reduced to a value
  - Parameters ARE evaluated before they are used
- NO reductions inside Abstractions
  - Abstractions are values
- eager, strict

$$\begin{aligned} & (\lambda x.x) ((\lambda x.x) (\lambda z.(\lambda x.x) z)) \\ \rightarrow & (\lambda x.x) (\lambda z.(\lambda x.x) z) \\ \rightarrow & \lambda z.(\lambda x.x) z \end{aligned}$$

## Call-by-Value

- Outer-most, only if right-hand side was reduced to a value
  - Parameters ARE evaluated before they are used
- NO reductions inside Abstractions
  - Abstractions are values
- eager, strict

$$\begin{aligned} & (\lambda x.x) ((\lambda x.x) (\lambda z.(\lambda x.x) z)) \\ \rightarrow & (\lambda x.x) (\lambda z.(\lambda x.x) z) \\ \rightarrow & \lambda z.(\lambda x.x) z \\ \not\rightarrow & \end{aligned}$$

## Call-by-Value

- Outer-most, only if right-hand side was reduced to a value
  - Parameters ARE evaluated before they are used
- NO reductions inside Abstractions
  - Abstractions are values
- eager, strict



# Church Encodings

---

- Encode Data into the Lambda Calculus
- To simplify our formulas, let's say that we have declarations

$$id \equiv \lambda x.x$$

$$id\ y \rightarrow y$$



# Definitions

$true \equiv$	$\lambda t. \lambda f. t$
$false \equiv$	$\lambda t. \lambda f. f$

# Definitions

$true \equiv \lambda t. \lambda f. t$

$false \equiv \lambda t. \lambda f. f$

$test \equiv \lambda c. \lambda t. \lambda f. c \ t \ f$

## Example

## Example

## Example

*test true a b*



## Example

*test true a b*

## Example

*test true a b*

$\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ true \ a \ b$

## Example

*test true a b*  
 $\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ \text{true} \ a \ b$

## Example

*test true a b*  
 $\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ \text{true} \ a \ b$   
 $\rightarrow (\lambda t. \lambda f. \text{true} \ t \ f) \ a \ b$

## Example

$test\ true\ a\ b$   
 $\equiv (\lambda c.\lambda t.\lambda f.c\ t\ f)\ true\ a\ b$   
 $\rightarrow (\lambda t.\lambda f.true\ t\ f)\ a\ b$

## Example

*test true a b*  
 $\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ \text{true} \ a \ b$   
 $\rightarrow (\lambda t. \lambda f. \text{true} \ t \ f) \ a \ b$   
 $\rightarrow (\lambda f. \text{true} \ a \ f) \ b$

## Example

$\text{test } \text{true } a \ b$   
 $\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ \text{true } a \ b$   
 $\rightarrow (\lambda t. \lambda f. \text{true } t \ f) \ a \ b$   
 $\rightarrow (\lambda f. \text{true } a \ f) \ b$

## Example

*test true a b*  
 $\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ \text{true} \ a \ b$   
 $\rightarrow (\lambda t. \lambda f. \text{true} \ t \ f) \ a \ b$   
 $\rightarrow (\lambda f. \text{true} \ a \ f) \ b$   
 $\rightarrow \text{true} \ a \ b$



## Example

$\text{test } \text{true } a \ b$   
 $\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ \text{true } a \ b$   
 $\rightarrow (\lambda t. \lambda f. \text{true } t \ f) \ a \ b$   
 $\rightarrow (\lambda f. \text{true } a \ f) \ b$   
 $\rightarrow \text{true } a \ b$

## Example

$\text{test } \text{true } a \ b$   
 $\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ \text{true } a \ b$   
 $\rightarrow (\lambda t. \lambda f. \text{true } t \ f) \ a \ b$   
 $\rightarrow (\lambda f. \text{true } a \ f) \ b$   
 $\rightarrow \text{true } a \ b$   
 $\equiv (\lambda t. \lambda f. t) \ a \ b$

## Example

$\text{test } \text{true } a \ b$   
 $\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ \text{true } a \ b$   
 $\rightarrow (\lambda t. \lambda f. \text{true } t \ f) \ a \ b$   
 $\rightarrow (\lambda f. \text{true } a \ f) \ b$   
 $\rightarrow \text{true } a \ b$   
 $\equiv (\lambda t. \lambda f. t) \ a \ b$

## Example

$\text{test } \text{true } a \ b$   
 $\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ \text{true } a \ b$   
 $\rightarrow (\lambda t. \lambda f. \text{true } t \ f) \ a \ b$   
 $\rightarrow (\lambda f. \text{true } a \ f) \ b$   
 $\rightarrow \text{true } a \ b$   
 $\equiv (\lambda t. \lambda f. t) \ a \ b$   
 $\rightarrow (\lambda f. a) \ b$

## Example

$\text{test } \text{true } a \ b$   
 $\equiv (\lambda c. \lambda t. \lambda f. c \ t \ f) \ \text{true } a \ b$   
 $\rightarrow (\lambda t. \lambda f. \text{true } t \ f) \ a \ b$   
 $\rightarrow (\lambda f. \text{true } a \ f) \ b$   
 $\rightarrow \text{true } a \ b$   
 $\equiv (\lambda t. \lambda f. t) \ a \ b$   
 $\rightarrow (\lambda f. a) \ b$

## Example

$test\ true\ a\ b$   
 $\equiv (\lambda c.\lambda t.\lambda f.c\ t\ f)\ true\ a\ b$   
 $\rightarrow (\lambda t.\lambda f.true\ t\ f)\ a\ b$   
 $\rightarrow (\lambda f.true\ a\ f)\ b$   
 $\rightarrow true\ a\ b$   
 $\equiv (\lambda t.\lambda f.t)\ a\ b$   
 $\rightarrow (\lambda f.a)\ b$   
 $\rightarrow a$

And

# Definitions

*true*  $\equiv$

$\lambda t.\lambda f.t$

*false*  $\equiv$

$\lambda t.\lambda f.f$



# Definitions

$true \equiv \lambda t. \lambda f. t$

$false \equiv \lambda t. \lambda f. f$

$and \equiv \lambda p. \lambda q. p \ q \ p$

## Example

## Example

## Example

*and true false*

## Example

*and true false*

## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \text{ true false}$

## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \text{ true } \text{false}$

## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \text{ true } \text{false}$   
 $\rightarrow (\lambda q. \text{true } q \ \text{true}) \text{ false}$



## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \ true \ false$   
 $\rightarrow (\lambda q. true \ q \ true) \ false$

## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \ true \ false$   
 $\rightarrow (\lambda q. true \ q \ true) \ false$   
 $\rightarrow true \ false \ true$

## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \text{ true } \text{false}$   
 $\rightarrow (\lambda q. \text{true } q \ \text{true}) \text{ false}$   
 $\rightarrow \text{true } \text{false } \text{true}$

## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \ \text{true} \ \text{false}$   
 $\rightarrow (\lambda q. \text{true} \ q \ \text{true}) \ \text{false}$   
 $\rightarrow \text{true} \ \text{false} \ \text{true}$   
 $\equiv (\lambda t. \lambda f. t) \ \text{false} \ \text{true}$

## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \ \text{true} \ \text{false}$   
 $\rightarrow (\lambda q. \text{true} \ q \ \text{true}) \ \text{false}$   
 $\rightarrow \text{true} \ \text{false} \ \text{true}$   
 $\equiv (\lambda t. \lambda f. t) \ \text{false} \ \text{true}$

## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \ \text{true} \ \text{false}$   
 $\rightarrow (\lambda q. \text{true} \ q \ \text{true}) \ \text{false}$   
 $\rightarrow \text{true} \ \text{false} \ \text{true}$   
 $\equiv (\lambda t. \lambda f. t) \ \text{false} \ \text{true}$   
 $\rightarrow (\lambda f. \text{false}) \ \text{true}$

## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \ \text{true} \ \text{false}$   
 $\rightarrow (\lambda q. \text{true} \ q \ \text{true}) \ \text{false}$   
 $\rightarrow \text{true} \ \text{false} \ \text{true}$   
 $\equiv (\lambda t. \lambda f. t) \ \text{false} \ \text{true}$   
 $\rightarrow (\lambda f. \text{false}) \ \text{true}$

## Example

*and true false*  
 $\equiv (\lambda p. \lambda q. p \ q \ p) \ \text{true} \ \text{false}$   
 $\rightarrow (\lambda q. \text{true} \ q \ \text{true}) \ \text{false}$   
 $\rightarrow \text{true} \ \text{false} \ \text{true}$   
 $\equiv (\lambda t. \lambda f. t) \ \text{false} \ \text{true}$   
 $\rightarrow (\lambda f. \text{false}) \ \text{true}$   
 $\rightarrow \text{false}$



Or

$\lambda p. \lambda q. p \ p \ q$



$$\textit{pair} \equiv \lambda x. \lambda y. \lambda z. z \ x \ y$$

# Definitions

*pair*  $\equiv$   $\lambda x.\lambda y.\lambda z.z\ x\ y$

*first*  $\equiv$   $(\lambda p.p)\ \lambda x.\lambda y.x$

*second*  $\equiv$   $(\lambda p.p)\ \lambda x.\lambda y.y$

## Example

## Example

## Example

$pair_{AB} \equiv$

$pair\ a\ b$

## Example

$pair_{AB} \equiv$

*pair*  $a\ b$



## Example

$$\begin{aligned} pair_{AB} &\equiv pair\ a\ b \\ &\equiv (\lambda x.\lambda y.\lambda z.z\ x\ y)\ a\ b \end{aligned}$$

## Example

$$\begin{aligned} pair_{AB} &\equiv pair\ a\ b \\ &\equiv (\lambda x.\lambda y.\lambda z.z\ x\ y)\ a\ b \end{aligned}$$

# Example

$$\begin{aligned} pair_{AB} &\equiv pair\ a\ b \\ &\equiv (\lambda x.\lambda y.\lambda z.z\ x\ y)\ a\ b \\ &\rightarrow (\lambda y.\lambda z.z\ a\ y)b \end{aligned}$$

# Example

$$\begin{aligned} pair_{AB} &\equiv pair\ a\ b \\ &\equiv (\lambda x.\lambda y.\lambda z.z\ x\ y)\ a\ b \\ &\rightarrow (\lambda y.\lambda z.z\ a\ y)b \end{aligned}$$

# Example

$$\begin{aligned} pair_{AB} &\equiv pair\ a\ b \\ &\equiv (\lambda x.\lambda y.\lambda z.z\ x\ y)\ a\ b \\ &\rightarrow (\lambda y.\lambda z.z\ a\ y)b \\ &\rightarrow \lambda z.z\ a\ b \end{aligned}$$

# Example

$$\begin{aligned} pair_{AB} &\equiv pair\ a\ b \\ &\equiv (\lambda x.\lambda y.\lambda z.z\ x\ y)\ a\ b \\ &\rightarrow (\lambda y.\lambda z.z\ a\ y)b \\ &\rightarrow \lambda z.z\ a\ b \\ &\equiv pair'_{ab} \end{aligned}$$

## Pairs (continued)

# Definitions

$pair \equiv \lambda x. \lambda y. \lambda z. z \ x \ y$

$first \equiv (\lambda p. p) \ \lambda x. \lambda y. x$

$pair'_{ab} \equiv \lambda z. z \ a \ b$



## Example

## Example

*first pair'*<sub>ab</sub>

# Example

*first*  $pair'_{ab}$

## Example

$$\equiv \text{first } pair'_{ab} \\ (\lambda p.p) (\lambda x.\lambda y.x) pair'_{ab}$$

## Example

$$\equiv \text{first } pair'_{ab} \\ (\lambda p.p) (\lambda x.\lambda y.x) pair'_{ab}$$

## Example

$$\begin{aligned} & \text{first } pair'_{ab} \\ \equiv & (\lambda p.p) (\lambda x.\lambda y.x) pair'_{ab} \\ \rightarrow & pair'_{ab} (\lambda x.\lambda y.x) \end{aligned}$$

## Example

$$\begin{aligned} & \text{first } \textit{pair}'_{ab} \\ \equiv & (\lambda p.p) (\lambda x.\lambda y.x) \textit{pair}'_{ab} \\ \rightarrow & \textit{pair}'_{ab} (\lambda x.\lambda y.x) \end{aligned}$$

## Example

$$\begin{aligned} & \text{first } \textit{pair}'_{ab} \\ \equiv & (\lambda p.p) (\lambda x.\lambda y.x) \textit{pair}'_{ab} \\ \rightarrow & \textit{pair}'_{ab} (\lambda x.\lambda y.x) \\ \equiv & (\lambda z.z \ a \ b) (\lambda x.\lambda y.x) \end{aligned}$$



## Example

$$\begin{aligned} & \text{first } \text{pair}'_{ab} \\ \equiv & (\lambda p.p) (\lambda x.\lambda y.x) \text{pair}'_{ab} \\ \rightarrow & \text{pair}'_{ab} (\lambda x.\lambda y.x) \\ \equiv & (\lambda z.z \ a \ b) (\lambda x.\lambda y.x) \end{aligned}$$

## Example

$$\begin{aligned} & \text{first } \text{pair}'_{ab} \\ \equiv & (\lambda p.p) (\lambda x.\lambda y.x) \text{pair}'_{ab} \\ \rightarrow & \text{pair}'_{ab} (\lambda x.\lambda y.x) \\ \equiv & (\lambda z.z \ a \ b) (\lambda x.\lambda y.x) \\ \rightarrow & (\lambda x.\lambda y.x) \ a \ b \end{aligned}$$

## Example

$$\begin{aligned} & \text{first } \text{pair}'_{ab} \\ \equiv & (\lambda p.p) (\lambda x.\lambda y.x) \text{pair}'_{ab} \\ \rightarrow & \text{pair}'_{ab} (\lambda x.\lambda y.x) \\ \equiv & (\lambda z.z \ a \ b) (\lambda x.\lambda y.x) \\ \rightarrow & (\lambda x.\lambda y.x) \ a \ b \end{aligned}$$

## Example

$$\begin{aligned} & \text{first } \text{pair}'_{ab} \\ \equiv & (\lambda p.p) (\lambda x.\lambda y.x) \text{pair}'_{ab} \\ \rightarrow & \text{pair}'_{ab} (\lambda x.\lambda y.x) \\ \equiv & (\lambda z.z \ a \ b) (\lambda x.\lambda y.x) \\ \rightarrow & (\lambda x.\lambda y.x) \ a \ b \\ \rightarrow & (\lambda y.a) \ b \end{aligned}$$

## Example

$$\begin{aligned} & \text{first } \text{pair}'_{ab} \\ \equiv & (\lambda p.p) (\lambda x.\lambda y.x) \text{pair}'_{ab} \\ \rightarrow & \text{pair}'_{ab} (\lambda x.\lambda y.x) \\ \equiv & (\lambda z.z \ a \ b) (\lambda x.\lambda y.x) \\ \rightarrow & (\lambda x.\lambda y.x) \ a \ b \\ \rightarrow & (\lambda y.a) \ b \end{aligned}$$

## Example

$$\begin{aligned} & \text{first } \text{pair}'_{ab} \\ \equiv & (\lambda p.p) (\lambda x.\lambda y.x) \text{pair}'_{ab} \\ \rightarrow & \text{pair}'_{ab} (\lambda x.\lambda y.x) \\ \equiv & (\lambda z.z \ a \ b) (\lambda x.\lambda y.x) \\ \rightarrow & (\lambda x.\lambda y.x) \ a \ b \\ \rightarrow & (\lambda y.a) \ b \\ \rightarrow & a \end{aligned}$$



# Peano axioms

Every natural number can be defined with 0 and a successor function

$$0 \equiv \lambda f. \lambda x. x$$

$$1 \equiv \lambda f. \lambda x. f \ x$$

$$2 \equiv \lambda f. \lambda x. f \ (f \ x)$$

$$3 \equiv \lambda f. \lambda x. f \ (f \ (f \ x))$$



0  $f$  is evaluated 0 times

1  $f$  is evaluated once

$x$  can be every lambda term

## Numerals Example - Successor

# Definitions

$$0 \equiv \lambda f. \lambda x. x$$

$$1 \equiv \lambda f. \lambda x. f \ x$$

# Definitions

$$0 \equiv \lambda f. \lambda x. x$$

$$1 \equiv \lambda f. \lambda x. f \ x$$

$$\textit{successor} \equiv \lambda n. \lambda f. \lambda x. f \ (n \ f \ x)$$

## Example

## Example

## Example

*successor 1*

# Example

*successor* 1



## Example

$\equiv$  *successor* 1  
 $(\lambda n. \lambda f. \lambda x. f (n f x)) 1$

## Example

$\equiv$  *successor* 1  
 $(\lambda n. \lambda f. \lambda x. f (n f x)) 1$

## Example

$\equiv$  *successor* 1  
 $\rightarrow$   $(\lambda n. \lambda f. \lambda x. f (n f x)) 1$   
 $\lambda f. \lambda x. f (1 f x)$

## Example

$\equiv$   $\text{successor } 1$   
 $\rightarrow (\lambda n. \lambda f. \lambda x. f (n f x)) 1$   
 $\rightarrow \lambda f. \lambda x. f (1 f x)$

## Example

$\equiv$   $\text{successor } 1$   
 $\rightarrow$   $(\lambda n. \lambda f. \lambda x. f (n f x)) 1$   
 $\equiv$   $\lambda f. \lambda x. f ((\lambda f. \lambda x. f x) f x)$

## Example

$\equiv$   $\text{successor } 1$   
 $\rightarrow (\lambda n. \lambda f. \lambda x. f (n f x)) 1$   
 $\rightarrow \lambda f. \lambda x. f (1 f x)$   
 $\equiv \lambda f. \lambda x. f ((\lambda f. \lambda x. f x) f x)$

## Example

$\text{successor } 1$

$\equiv (\lambda n. \lambda f. \lambda x. f (n f x)) 1$

$\rightarrow \lambda f. \lambda x. f (1 f x)$

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

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

## Example

$\text{successor } 1$

$\equiv (\lambda n. \lambda f. \lambda x. f (n f x)) 1$

$\rightarrow \lambda f. \lambda x. f (1 f x)$

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

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



## Example

$\text{successor } 1$

$\equiv (\lambda n. \lambda f. \lambda x. f (n f x)) 1$

$\rightarrow \lambda f. \lambda x. f (1 f x)$

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

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

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

## Example

$\text{successor } 1$

$\equiv (\lambda n. \lambda f. \lambda x. f (n f x)) 1$

$\rightarrow \lambda f. \lambda x. f (1 f x)$

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

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

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

$\equiv 2$

We use *Normal Order Reduction* to reduce inside abstractions!

## Numerals Example - $0 + 0$

$0 \equiv$

$\lambda f.\lambda x.x$

$$0 \equiv \lambda f. \lambda x. x$$

$$plus \equiv \lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)$$

*plus* 0 0

*plus* 0 0



$$\equiv \text{plus } 0 \ 0$$
$$\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0$$

$$\equiv \text{plus } 0 \ 0$$
$$\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & \quad (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & \quad (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \end{aligned}$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & \quad (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & \quad (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \end{aligned}$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & \quad (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & \quad (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \\ \rightarrow & \quad \lambda f. \lambda x. 0 \ f \ (0 \ f \ x) \end{aligned}$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & \quad (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & \quad (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \\ \rightarrow & \quad \lambda f. \lambda x. 0 \ f \ (0 \ f \ x) \end{aligned}$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & \quad (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & \quad (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \\ \rightarrow & \quad \lambda f. \lambda x. 0 \ f \ (0 \ f \ x) \\ \equiv & \quad \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (0 \ f \ x) \end{aligned}$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & \quad (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & \quad (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \\ \rightarrow & \quad \lambda f. \lambda x. 0 \ f \ (0 \ f \ x) \\ \equiv & \quad \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (0 \ f \ x) \end{aligned}$$



$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \\ \rightarrow & \lambda f. \lambda x. 0 \ f \ (0 \ f \ x) \\ \equiv & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (0 \ f \ x) \\ \rightarrow & \lambda f. \lambda x. (\lambda x. x) \ (0 \ f \ x) \end{aligned}$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & \quad (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & \quad (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \\ \rightarrow & \quad \lambda f. \lambda x. 0 \ f \ (0 \ f \ x) \\ \equiv & \quad \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (0 \ f \ x) \\ \rightarrow & \quad \lambda f. \lambda x. (\lambda x. x) \ (0 \ f \ x) \end{aligned}$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \\ \rightarrow & \lambda f. \lambda x. 0 \ f \ (0 \ f \ x) \\ \equiv & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (0 \ f \ x) \\ \rightarrow & \lambda f. \lambda x. (\lambda x. x) \ (0 \ f \ x) \\ \rightarrow & \lambda f. \lambda x. 0 \ f \ x \end{aligned}$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \\ \rightarrow & \lambda f. \lambda x. 0 \ f \ (0 \ f \ x) \\ \equiv & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (0 \ f \ x) \\ \rightarrow & \lambda f. \lambda x. (\lambda x. x) \ (0 \ f \ x) \\ \rightarrow & \lambda f. \lambda x. 0 \ f \ x \end{aligned}$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \\ \rightarrow & \lambda f. \lambda x. 0 \ f \ (0 \ f \ x) \\ \equiv & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (0 \ f \ x) \\ \rightarrow & \lambda f. \lambda x. (\lambda x. x) \ (0 \ f \ x) \\ \rightarrow & \lambda f. \lambda x. 0 \ f \ x \\ \equiv & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ x \end{aligned}$$

$$\begin{aligned} & \text{plus } 0 \ 0 \\ \equiv & (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ 0 \ 0 \\ \rightarrow & (\lambda n. \lambda f. \lambda x. 0 \ f \ (n \ f \ x)) \ 0 \\ \rightarrow & \lambda f. \lambda x. 0 \ f \ (0 \ f \ x) \\ \equiv & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (0 \ f \ x) \\ \rightarrow & \lambda f. \lambda x. (\lambda x. x) \ (0 \ f \ x) \\ \rightarrow & \lambda f. \lambda x. 0 \ f \ x \\ \equiv & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ x \end{aligned}$$

$plus\ 0\ 0$

$\equiv (\lambda m.\lambda n.\lambda f.\lambda x.m\ f\ (n\ f\ x))\ 0\ 0$

$\rightarrow (\lambda n.\lambda f.\lambda x.0\ f\ (n\ f\ x))\ 0$

$\rightarrow \lambda f.\lambda x.0\ f\ (0\ f\ x)$

$\equiv \lambda f.\lambda x.(\lambda f.\lambda x.x)\ f\ (0\ f\ x)$

$\rightarrow \lambda f.\lambda x.(\lambda x.x)\ (0\ f\ x)$

$\rightarrow \lambda f.\lambda x.0\ f\ x$

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

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

$plus\ 0\ 0$

$\equiv (\lambda m.\lambda n.\lambda f.\lambda x.m\ f\ (n\ f\ x))\ 0\ 0$

$\rightarrow (\lambda n.\lambda f.\lambda x.0\ f\ (n\ f\ x))\ 0$

$\rightarrow \lambda f.\lambda x.0\ f\ (0\ f\ x)$

$\equiv \lambda f.\lambda x.(\lambda f.\lambda x.x)\ f\ (0\ f\ x)$

$\rightarrow \lambda f.\lambda x.(\lambda x.x)\ (0\ f\ x)$

$\rightarrow \lambda f.\lambda x.0\ f\ x$

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

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



$plus\ 0\ 0$

$\equiv (\lambda m.\lambda n.\lambda f.\lambda x.m\ f\ (n\ f\ x))\ 0\ 0$

$\rightarrow (\lambda n.\lambda f.\lambda x.0\ f\ (n\ f\ x))\ 0$

$\rightarrow \lambda f.\lambda x.0\ f\ (0\ f\ x)$

$\equiv \lambda f.\lambda x.(\lambda f.\lambda x.x)\ f\ (0\ f\ x)$

$\rightarrow \lambda f.\lambda x.(\lambda x.x)\ (0\ f\ x)$

$\rightarrow \lambda f.\lambda x.0\ f\ x$

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

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

$\rightarrow \lambda f.\lambda x.x$

$plus\ 0\ 0$

$\equiv (\lambda m.\lambda n.\lambda f.\lambda x.m\ f\ (n\ f\ x))\ 0\ 0$

$\rightarrow (\lambda n.\lambda f.\lambda x.0\ f\ (n\ f\ x))\ 0$

$\rightarrow \lambda f.\lambda x.0\ f\ (0\ f\ x)$

$\equiv \lambda f.\lambda x.(\lambda f.\lambda x.x)\ f\ (0\ f\ x)$

$\rightarrow \lambda f.\lambda x.(\lambda x.x)\ (0\ f\ x)$

$\rightarrow \lambda f.\lambda x.0\ f\ x$

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

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

$\rightarrow \lambda f.\lambda x.x$

$\equiv 0$

**(Almost) The End**

---

## Good Math

- Easy to understand
- Blog: <http://www.goodmath.org/blog/>

## The Implementation of Functional Programming Languages

- How to compile to the Lambda Calculus?
- Free: <https://www.microsoft.com/en-us/research/publication/the-implementation-of-functional-programming-languages/>

## Types and Programming Languages

- Types systems explained by building interpreters and proving attributes
- Very "mathematical", but very complete

**Thanks**

---

- Hope you enjoyed this talk and learned something new.
- Hope it wasn't too much math and dusty formulas ... :)