

# Untyped Lambda Calculus

Structure and Evaluation, Currying, Church Encodings

---

Sven Tennie

November 17, 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

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

# 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

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

**Example** -  $(\lambda x. \lambda y. x y) a b$

## **Abstractions**

Think: Function Definitions

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

**Example** -  $(\lambda x. \lambda y. x \ y) \ a \ b$

## **Abstractions**

Think: Function Definitions

$(\lambda x. \lambda y. \underline{x \ y}) \ a \ b$

## **Variables**

Think: Parameters

$(\lambda x. \lambda y. \underline{x} \ \underline{y}) \ \underline{a} \ \underline{b}$

## Example - $(\lambda x. \lambda y. x \ y) \ a \ b$

### Abstractions

Think: Function Definitions

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

### Variables

Think: Parameters

$(\lambda x. \lambda y. \underline{x} \ \underline{y}) \ \underline{a} \ \underline{b}$

### Applications

Think: Function Calls

$(\lambda x. \lambda y. x \ y)$   $\underline{a} \ \underline{b}$

**Example** -  $(\lambda x. \lambda y. x \ y) \ a \ b$

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

**Example** -  $(\lambda x. \lambda y. x \ y) \ a \ b$

$(\lambda x. \quad \lambda y. x \quad y) \quad a \quad b$       Substitute  $x \mapsto a$

**Example** -  $(\lambda x. \lambda y. x y) a b$

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



## Example - $(\lambda x. \lambda y. x y) a b$

$$\begin{array}{llll} (\lambda x. & \lambda y. x & y) & a \quad b \quad \text{Substitute } x \mapsto a \\ \rightarrow & (\lambda y. a & y) & b \end{array}$$

## Example - $(\lambda x. \lambda y. x \ y) \ a \ b$

	$(\lambda x. \lambda y. x \ y)$	$a$	$b$	Substitute $x \mapsto a$
$\rightarrow$	$(\lambda y. a \ y)$		$b$	Substitute $y \mapsto b$

## Example - $(\lambda x. \lambda y. x \ y) \ a \ b$

$(\lambda x. \lambda y. x \ y) \ a \ b$       Substitute  $x \mapsto a$   
 $\rightarrow (\lambda y. a \ y) \ b$       Substitute  $y \mapsto b$

## Example - $(\lambda x. \lambda y. x \ y) \ a \ b$

$(\lambda x. \lambda y. x \ y) \ a \ b$       Substitute  $x \mapsto a$   
 $\rightarrow (\lambda y. a \ y) \ b$       Substitute  $y \mapsto b$   
 $\rightarrow a \ b$

# 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*

$\lambda x, \lambda y$  *binders*

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

## ***Bound and Free***

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

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

$\lambda x, \lambda y$  *binders*

## **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 function with  $n$  arguments
- Create a function that takes one argument and returns a function with  $n - 1$  arguments

## Idea

- Take a function with  $n$  arguments
- Create a function 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 function with  $n$  arguments
- Create a function 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  $f$ :

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

# 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  $f$ ,  $\eta$  conversion not possible:

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

- 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

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

# Full Beta-Reduction

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

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

## Full Beta-Reduction

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

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

## Normal Order Reduction

- Left-most, Outer-most RedEx

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

## Normal Order Reduction

- Left-most, Outer-most RedEx

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

## Normal Order Reduction

- Left-most, Outer-most RedEx

# Normal Order Reduction

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

## Normal Order Reduction

- Left-most, Outer-most RedEx



# Normal Order Reduction

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

## Normal Order Reduction

- Left-most, Outer-most RedEx

# Normal Order Reduction

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

## Normal Order Reduction

- Left-most, Outer-most RedEx

# Normal Order Reduction

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

## Normal Order Reduction

- Left-most, Outer-most RedEx

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

## 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 results  $\rightarrow$  *Call-by-Need*

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

## 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 results  $\rightarrow$  *Call-by-Need*

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

## 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 results  $\rightarrow$  *Call-by-Need*

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

## 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 results  $\rightarrow$  *Call-by-Need*

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

## 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 results  $\rightarrow$  *Call-by-Need*



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

## 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 results  $\rightarrow$  *Call-by-Need*

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

## Call-by-Value

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

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

## Call-by-Value

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

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

## Call-by-Value

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

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

## Call-by-Value

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

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

## Call-by-Value

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

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

## Call-by-Value

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

# Church Encodings

---



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

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

$$id\ y \rightarrow y$$

# Booleans

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

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

# Booleans

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

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

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

*test true a b*

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

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

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

# Booleans

*test true a b*

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

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

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

# Booleans

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

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

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

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

# Booleans

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

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

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

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

# Booleans

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

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

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

$test \ true \ a \ b$

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

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



# Booleans

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

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

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

$test \ true \ a \ b$

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

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

# Booleans

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

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

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

$test \ true \ a \ b$

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

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

$\rightarrow (\lambda f. true \ a \ f) \ b$

# Booleans

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

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

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

$test \ true \ a \ b$

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

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

$\rightarrow (\lambda f. true \ a \ f) \ b$

# Booleans

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

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

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

$test \ true \ a \ b$

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

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

$\rightarrow (\lambda f. true \ a \ f) \ b$

$\rightarrow true \ a \ b$

# Booleans

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

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

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

$test \ true \ a \ b$

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

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

$\rightarrow (\lambda f. true \ a \ f) \ b$

$\rightarrow true \ a \ b$

# Booleans

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

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

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

$test \ true \ a \ b$

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

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

$\rightarrow (\lambda f. true \ a \ f) \ b$

$\rightarrow true \ a \ b$

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

# Booleans

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

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

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

$test \ true \ a \ b$

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

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

$\rightarrow (\lambda f. true \ a \ f) \ b$

$\rightarrow true \ a \ b$

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

# Booleans

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

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

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

$test \ true \ a \ b$

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

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

$\rightarrow (\lambda f. true \ a \ f) \ b$

$\rightarrow true \ a \ b$

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

$\rightarrow (\lambda f. a) \ b$



# Booleans

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

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

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

$test \ true \ a \ b$

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

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

$\rightarrow (\lambda f. true \ a \ f) \ b$

$\rightarrow true \ a \ b$

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

$\rightarrow (\lambda f. a) \ b$

# Booleans

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

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

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

$test \ true \ a \ b$

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

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

$\rightarrow (\lambda f. true \ a \ f) \ b$

$\rightarrow true \ a \ b$

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

$\rightarrow (\lambda f. a) \ b$

$\rightarrow a$

# And

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

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

# And

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

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

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

# And

*and true false*

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

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

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

# And

*and true false*

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

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

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

# And

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

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

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

*and true false*

$\equiv (\lambda p. \lambda q. p \ q \ p) \ true \ false$

# And

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

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

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

*and true false*

$\equiv (\lambda p. \lambda q. p \ q \ p) \ i \ o$



# And

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

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

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

*and true false*

$\equiv (\lambda p. \lambda q. p \ q \ p) \ i\!rue \ f\!alse$

$\rightarrow (\lambda q. i\!rue \ q \ i\!rue) \ f\!alse$

# And

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

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

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

*and true false*

$\equiv (\lambda p. \lambda q. p \ q \ p) \ i \ o$

$\rightarrow (\lambda q. i \ q \ i) \ o$

# And

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

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

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

*and true false*

$\equiv (\lambda p.\lambda q.p\ q\ p)\ true\ false$

$\rightarrow (\lambda q.true\ q\ true)\ false$

$\rightarrow true\ false\ true$

# And

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

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

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

$and \ true \ false$

$\equiv (\lambda p. \lambda q. p \ q \ p) \ true \ false$

$\rightarrow (\lambda q. true \ q \ true) \ false$

$\rightarrow true \ false \ true$

# And

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

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

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

*and true false*

$\equiv (\lambda p.\lambda q.p\ q\ p)\ true\ false$

$\rightarrow (\lambda q.true\ q\ true)\ false$

$\rightarrow true\ false\ true$

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

# And

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

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

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

*and true false*

$\equiv (\lambda p.\lambda q.p\ q\ p)\ true\ false$

$\rightarrow (\lambda q.true\ q\ true)\ false$

$\rightarrow true\ false\ true$

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

# And

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

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

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

*and true false*

$\equiv (\lambda p.\lambda q.p \ q \ p) \ true \ false$

$\rightarrow (\lambda q.true \ q \ true) \ false$

$\rightarrow true \ false \ true$

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

$\rightarrow (\lambda f.false) \ true$

# And

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

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

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

$and \ true \ false$

$\equiv (\lambda p. \lambda q. p \ q \ p) \ true \ false$

$\rightarrow (\lambda q. true \ q \ true) \ false$

$\rightarrow true \ false \ true$

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

$\rightarrow (\lambda f. false) \ true$



# And

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

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

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

*and true false*

$\equiv (\lambda p.\lambda q.p\ q\ p)\ true\ false$

$\rightarrow (\lambda q.true\ q\ true)\ false$

$\rightarrow true\ false\ true$

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

$\rightarrow (\lambda f.false)\ true$

$\rightarrow false$

Or

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

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

# Pairs

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

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

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

# Pairs

$pair_{AB} \equiv$

$pair\ a\ b$

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

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

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

# Pairs

$pair_{AB} \equiv$

*pair* *a b*

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

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

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

# Pairs

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

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

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

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

# Pairs

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

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

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

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



# Pairs

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

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

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

$pair_{AB} \equiv$   $pair \ a \ b$

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

$\rightarrow$   $(\lambda y. \lambda z. z \ a \ y) \ b$

# Pairs

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

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

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

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

# Pairs

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

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

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

$pair_{AB} \equiv$  *pair*  $a\ b$

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

$\rightarrow (\lambda y.\lambda z.z\ a\ y)\ b$

$\rightarrow \lambda z.z\ a\ b$

# Pairs

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

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

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

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

## Pairs (continued)

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

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

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

## Pairs (continued)

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

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

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

*pair'*<sub>ab</sub>  $\equiv \lambda z. z \ a \ b$

## Pairs (continued)

*first*  $pair'_{ab}$

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

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

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

## Pairs (continued)

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



## Pairs (continued)

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

## Pairs (continued)

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

## Pairs (continued)

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

## Pairs (continued)

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

## Pairs (continued)

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

## Pairs (continued)

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

## Pairs (continued)

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

## Pairs (continued)

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



## Pairs (continued)

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

## Pairs (continued)

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

# Numerals

## Peano Axioms

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

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

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

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

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

## Meaning

0  $f$  is evaluated 0 times

1  $f$  is evaluated once

$x$  can be every lambda term

## Numerals Example - Successor

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

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

## Numerals Example - Successor

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

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

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

## Numerals Example - Successor

*successor* 1

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

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

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

## Numerals Example - Successor

*successor* 1

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

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

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

## Numerals Example - Successor

$$\begin{aligned} 0 &\equiv \lambda f. \lambda x. x && \equiv \text{successor } 1 \\ 1 &\equiv \lambda f. \lambda x. f \, x && \equiv (\lambda n. \lambda f. \lambda x. f \, (n \, f \, x)) \, 1 \end{aligned}$$

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



## Numerals Example - Successor

$$\begin{aligned} 0 &\equiv \lambda f. \lambda x. x && \equiv \text{successor } 1 \\ 1 &\equiv \lambda f. \lambda x. f \, x && \equiv (\lambda n. \lambda f. \lambda x. f \, (n \, f \, x)) \, 1 \end{aligned}$$

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

## Numerals Example - Successor

$$\begin{array}{llll} & & & \text{successor } 1 \\ 0 \equiv & \lambda f. \lambda x. x & \equiv & (\lambda n. \lambda f. \lambda x. f (n f x)) 1 \\ 1 \equiv & \lambda f. \lambda x. f x & \rightarrow & \lambda f. \lambda x. f (1 f x) \end{array}$$

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

## Numerals Example - Successor

$$\begin{array}{llll} & & & \text{successor } 1 \\ 0 \equiv & \lambda f. \lambda x. x & \equiv & (\lambda n. \lambda f. \lambda x. f (n f x)) 1 \\ 1 \equiv & \lambda f. \lambda x. f x & \rightarrow & \lambda f. \lambda x. f (1 f x) \end{array}$$

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

### Note

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

## Numerals Example - Successor

$$\begin{array}{llll} & & & \text{successor } 1 \\ 0 \equiv & \lambda f. \lambda x. x & \equiv & (\lambda n. \lambda f. \lambda x. f (n \ f \ x)) \ 1 \\ 1 \equiv & \lambda f. \lambda x. f \ x & \rightarrow & \lambda f. \lambda x. f \ (\textcolor{brown}{1} \ f \ x) \\ & & \equiv & \lambda f. \lambda x. f ((\lambda f. \lambda x. f \ x) \ f \ x) \end{array}$$

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

### Note

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

## Numerals Example - Successor

$$\begin{array}{llll} & & & \text{successor } 1 \\ 0 \equiv & \lambda f. \lambda x. x & \equiv & (\lambda n. \lambda f. \lambda x. f (n f x)) 1 \\ 1 \equiv & \lambda f. \lambda x. f x & \rightarrow & \lambda f. \lambda x. f (1 f x) \\ & & \equiv & \lambda f. \lambda x. f ((\lambda f. \lambda x. f x) f x) \end{array}$$

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

### Note

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

## Numerals Example - Successor

$$\begin{aligned} 0 &\equiv \lambda f. \lambda x. x && \equiv \text{successor } 1 \\ 1 &\equiv \lambda f. \lambda x. f \ x && \equiv (\lambda n. \lambda f. \lambda x. f \ (n \ f \ x)) \ 1 \\ &&& \rightarrow \lambda f. \lambda x. f \ (1 \ f \ x) \\ &&& \equiv \lambda f. \lambda x. f \ ((\lambda f. \lambda x. f \ x) \ f \ x) \\ &&& \rightarrow \lambda f. \lambda x. f \ ((\lambda x. f \ x) \ x) \\ \text{successor} &\equiv \lambda n. \lambda f. \lambda x. f \ (n \ f \ x) \end{aligned}$$

### Note

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

## Numerals Example - Successor

$$\begin{aligned} 0 &\equiv \lambda f. \lambda x. x && \equiv \text{successor } 1 \\ 1 &\equiv \lambda f. \lambda x. f \ x && \equiv (\lambda n. \lambda f. \lambda x. f \ (n \ f \ x)) \ 1 \\ &&& \rightarrow \lambda f. \lambda x. f \ (1 \ f \ x) \\ &&& \equiv \lambda f. \lambda x. f \ ((\lambda f. \lambda x. f \ x) \ f \ x) \\ &&& \rightarrow \lambda f. \lambda x. f \ ((\lambda x. f \ x) \ x) \\ \text{successor} &\equiv \lambda n. \lambda f. \lambda x. f \ (n \ f \ x) \end{aligned}$$

### Note

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

## Numerals Example - Successor

$$\begin{array}{llll} & & & \text{successor } 1 \\ 0 \equiv & \lambda f. \lambda x. x & \equiv & (\lambda n. \lambda f. \lambda x. f (n \ f \ x)) \ 1 \\ 1 \equiv & \lambda f. \lambda x. f \ x & \rightarrow & \lambda f. \lambda x. f (1 \ f \ x) \\ & & \equiv & \lambda f. \lambda x. f ((\lambda f. \lambda x. f \ x) \ f \ x) \\ \text{successor} \equiv & \lambda n. \lambda f. \lambda x. f (n \ f \ x) & \rightarrow & \lambda f. \lambda x. f ((\lambda x. f \ x) \ x) \\ & & \rightarrow & \lambda f. \lambda x. f (f \ x) \end{array}$$

### Note

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



## Numerals Example - Successor

$$\begin{array}{llll} & & & \text{successor } 1 \\ 0 \equiv & \lambda f. \lambda x. x & \equiv & (\lambda n. \lambda f. \lambda x. f (n \ f \ x)) \ 1 \\ 1 \equiv & \lambda f. \lambda x. f \ x & \rightarrow & \lambda f. \lambda x. f (1 \ f \ x) \\ & & \equiv & \lambda f. \lambda x. f ((\lambda f. \lambda x. f \ x) \ f \ x) \\ \text{successor} \equiv & \lambda n. \lambda f. \lambda x. f (n \ f \ x) & \rightarrow & \lambda f. \lambda x. f ((\lambda x. f \ x) \ x) \\ & & \rightarrow & \lambda f. \lambda x. f (f \ x) \\ & & \equiv & 2 \end{array}$$

### Note

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

## Numerals Example - $0 + 0$

$0 \equiv$

$\lambda f. \lambda x. x$

## Numerals Example - 0 + 0

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

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

## Numerals Example - 0 + 0

*plus* 0 0

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

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

## Numerals Example - 0 + 0

*plus* 0 0

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

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

## Numerals Example - $0 + 0$

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

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

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

## Numerals Example - $0 + 0$

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

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

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

## Numerals Example - 0 + 0

*plus* 0 0

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

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

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

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



## Numerals Example - 0 + 0

*plus* 0 0

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

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

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

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

## Numerals Example - $0 + 0$

*plus* 0 0

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

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

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

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

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

## Numerals Example - $0 + 0$

*plus* 0 0

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

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

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

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

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

## Numerals Example - $0 + 0$

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

## Numerals Example - $0 + 0$

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

## Numerals Example - $0 + 0$

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

## Numerals Example - $0 + 0$

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

# Numerals Example - 0 + 0

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



# Numerals Example - 0 + 0

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

# Numerals Example - $0 + 0$

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

# Numerals Example - $0 + 0$

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

# Numerals Example - 0 + 0

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

# Numerals Example - $0 + 0$

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

# Numerals Example - 0 + 0

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

# Numerals Example - 0 + 0

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

**End**

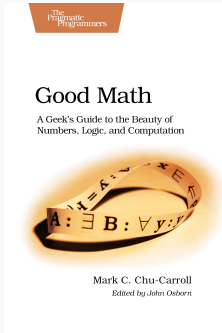
---



# Thanks

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

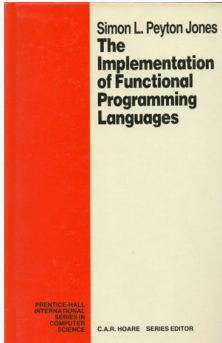
# Good Math



/"A Geek's Guide to the Beauty of  
Numbers, Logic, and Computation"/

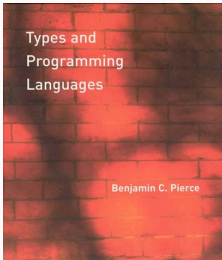
- Easy to understand

# The Implementation of Functional Programming Languages



- How to compile to the Lambda Calculus?
- Out-of-print, but freely available
  - <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 properties
- Very "mathematical", but very complete and self-contained