

# Lambda calculus

Functional models of computation

Ivan Trepakov

NSU Sys.Pro

## History

- 1928 - Hilbert's *Entscheidungsproblem* <sup>1</sup>
  - Is there an *algorithm* for deciding whether a proposition in first-order logic is true or false?
- Replacement for set theory as foundation of mathematics
  - 1930 - Combinatory logic (*Curry, Schönfinkel*)
  - 1932 -  $\lambda$ -calculus (*Church*)
  - 1935 - Kleene-Rosser paradox
- Effective computability
  - 1935 - Untyped  $\lambda$ -calculus (*Church, Kleene, Rosser*)
  - 1936 - Turing machine
  - 1936 - Church-Turing thesis
- 1936 - Undecidability of first-order logic
  - Halting problem of Turing machine
  - Equivalence of  $\lambda$ -terms

---

<sup>1</sup>German for “decision problem”

## History

- 1928 - Hilbert's *Entscheidungsproblem*<sup>1</sup>
  - Is there an *algorithm* for deciding whether a proposition in first-order logic is true or false?
- Replacement for set theory as foundation of mathematics
  - 1930 - Combinatory logic (*Curry, Schönfinkel*)
  - 1932 -  $\lambda$ -calculus (*Church*)
  - 1935 - Kleene-Rosser paradox
- Effective computability
  - 1935 - Untyped  $\lambda$ -calculus (*Church, Kleene, Rosser*)
  - 1936 - Turing machine
  - 1936 - Church-Turing thesis
- 1936 - Undecidability of first-order logic
  - Halting problem of Turing machine
  - Equivalence of  $\lambda$ -terms

---

<sup>1</sup>German for "decision problem"

## David Hilbert

- Haskell Curry
- Wilhelm Ackermann
- John von Neumann
- Ernst Zermelo
- ...

## Alonzo Church

- Stephen Cole Kleene
- J. Barkley Rosser
- Alan Turing
- Dana Scott
- Michael O. Rabin
- ...

## Grammar

$$term ::= \underbrace{var}_{\text{Variable}} \mid \underbrace{(term\ term)}_{\text{Application}} \mid \underbrace{(\lambda var. term)}_{\text{Abstraction}}$$

## Examples

$$\lambda x. x \quad (\lambda x. xx)(\lambda y. yy) \quad \lambda f. \lambda x. f(fx)$$

## Conventions

- Application is left associative  
 $abc = (ab)c$
- Abstraction is right associative  
 $\lambda x. \lambda y. x = \lambda x. (\lambda y. x)$
- Consecutive abstractions can be combined  
 $\lambda x. \lambda y. x = \lambda xy. x$

## Grammar

$$term ::= \underbrace{var}_{\text{Variable}} \mid \underbrace{(term \ term)}_{\text{Application}} \mid \underbrace{(\lambda var. term)}_{\text{Abstraction}}$$

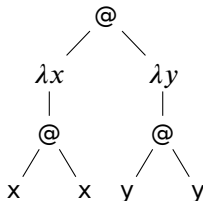
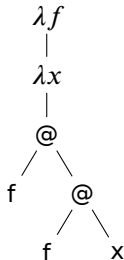
## Examples

 $\lambda x. x \quad (\lambda x. xx)(\lambda y. yy) \quad \lambda f. \lambda x. f(fx)$ 

## Conventions

- Application is left associative  
 $abc = (ab)c$
- Abstraction is right associative  
 $\lambda x. \lambda y. x = \lambda x. (\lambda y. x)$
- Consecutive abstractions can be combined  
 $\lambda x. \lambda y. x = \lambda xy. x$

## Tree representation

$$\begin{array}{c} \lambda x. x \\ | \\ \lambda x \\ | \\ x \end{array}$$
 $(\lambda x. xx)(\lambda y. yy)$ 

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


# Free and bound variables

## Free variables $FV(t)$

Variable:  $FV(x) = \{x\}$

Application:  $FV(MN) = FV(M) \cup FV(N)$

Abstraction:  $FV(\lambda x. M) = FV(M) \setminus \{x\}$

## Bound variables $BV(t)$

Variable:  $BV(x) = \emptyset$

Application:  $BV(MN) = BV(M) \cup BV(N)$

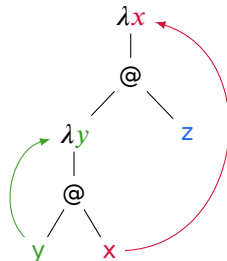
Abstraction:  $BV(\lambda x. M) = BV(M) \cup \{x\}$

## Closed terms

Term  $t$  is called **closed** or **combinator** if  $FV(t) = \emptyset$

## Example

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



# Substitution

Substitution  $t_{[v:=S]}$

$$x_{[v:=S]} = \begin{cases} S & v = x \\ x & v \neq x \end{cases}$$

$$(MN)_{[v:=S]} = (M_{[v:=S]} N_{[v:=S]})$$

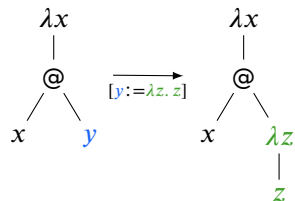
$$(\lambda x. M)_{[v:=S]} = \begin{cases} \lambda x. M & v = x \\ \lambda x. M_{[v:=S]} & v \neq x \end{cases}$$

Safe substitution

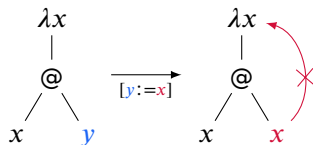
Substitution  $t_{[v:=S]}$  is **safe** if  $BV(t) \cap FV(S) = \emptyset$

Example

$$(\lambda x. xy)_{[y:=\lambda z. z]} = \lambda x. x(\lambda z. z)$$

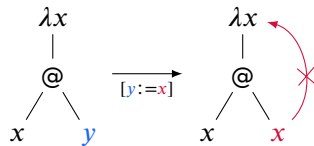


$$(\lambda x. xy)_{[y:=x]} = \lambda x. xx$$



## Problem

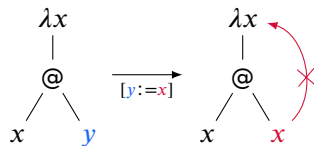
$$\lambda x. x y_{[y:=x]} = \lambda x. x x$$





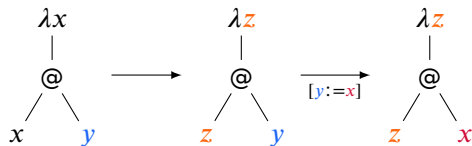
## Problem

$$\lambda x. x y_{[y:=x]} = \lambda x. x x$$



## Solution

$$(\lambda x. x y)_{[y:=x]} \equiv (\lambda z. z y)_{[y:=x]} = \lambda z. z x$$



# Renaming

## $\alpha$ -equivalence

$$\lambda x. M \equiv_{\alpha} \lambda y. M_{[x:=y]} \quad \text{if } x \notin FV(M)$$

$$\lambda x. M \equiv_{\alpha} \lambda x. N \quad \text{if } M \equiv_{\alpha} N$$

$$MP \equiv_{\alpha} NP \quad \text{if } M \equiv_{\alpha} N$$

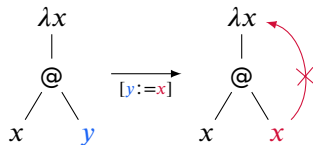
$$PM \equiv_{\alpha} PN \quad \text{if } M \equiv_{\alpha} N$$

## Conventions

- $\lambda$ -terms are considered **identical** up to  $\alpha$ -equivalence
- Appropriate renaming happens **implicitly** if required during substitution

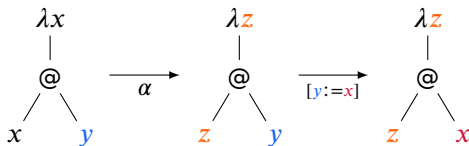
## Problem

$$\lambda x. xy_{[y:=x]} = \lambda x. xx$$



## Solution

$$(\lambda x. xy)_{[y:=x]} \equiv_{\alpha} (\lambda z. zy)_{[y:=x]} = \lambda z. zx$$



$\beta$ -conversion

$$(\lambda x. M)N \xleftrightarrow[\beta]{} M_{[x:=N]}$$

$\beta$ -reduction

$\beta$ -abstraction

$\eta$ -conversion

$$(\lambda x. Mx) \underset{\eta}{\longleftrightarrow} M \text{ if } x \notin FV(M)$$



# Normal order reduction

First Church-Rosser theorem

Second Church-Rosser theorem

Normal order reduction

Fixed-point combinator

Curry's  $Y$ -combinator

$$Y = \lambda f. (\lambda x. f(xx)) (\lambda x. f(xx))$$

Turing's  $\Theta$ -combinator

$$\Theta = (\lambda xy. x(xxy)) (\lambda xy. x(xxy))$$

Undecidability



Church numerals

Relation to folds

Algebraic data types

Predecessor

Q&A