

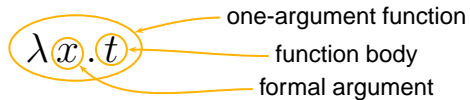
# G54FOP: Lecture 7

## The Untyped $\lambda$ -Calculus I: Introduction

Henrik Nilsson  
University of Nottingham, UK

### Key Idea

$\lambda$ -abstraction (or anonymous function):



### Exercise

In the following:

- Which variables are free and which are bound?
- Which terms are open and which are closed?

- (a)  $x$                       (d)  $\lambda x.\lambda y.x y$   
 (b)  $\lambda x.x$                 (e)  $(\lambda x.x) x$   
 (c)  $\lambda x.y$                 (f)  $\lambda x.\lambda y.(\lambda x.x y) (\lambda z.x y)$

### The $\lambda$ -Calculus: What is it? (1)

- Pure notion of effective computation procedure: **all** computation reduced to function definition and application.
- Invented in the 1920s by Alonzo Church.
- Cf. other formalisations of the notion of effective computation; e.g., the Turing machine.
- The  $\lambda$ -calculus and Turing Machines are equivalent in that they capture the exact same notion of what “computation” means.

### Syntax

$t \rightarrow$		terms:
	$x$	variable
	$\lambda x.t$	abstraction
	$t t$	application

Note:

- $x$  is the syntactic category of variables. We will use actual names like  $x, y, z, u, v, w, \dots$
- $\lambda$ -abstractions often named for convenience. E.g.  $I \equiv \lambda x.x$ . **Just an abbreviation!** So e.g.  $F \equiv \lambda x.(\dots F \dots)$  **not** valid def. Why?

### Operational Semantics (1)

**Sole** means of computation:  $\beta$ -reduction or **function application**:

$$(\lambda x.t_1) t_2 \rightarrow_{\beta} [x \mapsto t_2]t_1$$

where

$$[x \mapsto t_2]t_1$$

means “term  $t_1$  with all **free** occurrences of  $x$  (with respect to  $t_1$ ) replaced by  $t_2$ .”

Subtle problems concerning **name clashes** will be considered later.

### The $\lambda$ -Calculus: What is it? (2)

- The Church-Turing Hypothesis: The  $\lambda$ -calculus, Turing Machines, etc. coincides with our intuitive understanding of what “computation” means.
- The  $\lambda$ -calculus is important because it is at once:
  - very simple, yet in essence a practically useful programming language
  - mathematically precise, allowing for formal reasoning.

### Scope

- An **occurrence** of  $x$  is **bound** if it occurs in the body  $t$  of a  $\lambda$ -abstraction  $\lambda x.t$ .
- A non-bound occurrence is **free**.
- A  $\lambda$ -term with **no free** variables is called **closed**. Otherwise **open**.
- A closed  $\lambda$ -term is called a **combinator**.

### Operational Semantics (2)

A term that can be  $\beta$ -reduced is called a **( $\beta$ -)redex**.

Exercise: Underline the redexes in

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