G54FOP: Lecture 7

The Untyped λ-Calculus I: Introduction

Henrik Nilsson

University of Nottingham, UK

The λ-Calculus: What is it? (1)

The λ-Calculus: What is it? (1)

- Pure notion of effective computation procedure: ***all*** computation reduced to function definition and application.

The λ-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.

The λ-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 λ-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 λ-calculus and Turing Machines are equivalent in that they capture the exact same notion of what “computation” means.

The λ-Calculus: What is it? (2)

- The Church-Turing Hypothesis: The λ-calculus, Turing Machines, etc. coincides with our intuitive understanding of what "computation" means.

The λ-Calculus: What is it? (2)

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

Key Idea

λ-abstraction (or anonymous function):

$$\lambda\, x\, .\, t$$

Key Idea

λ-abstraction (or anonymous function):

$$\lambda x \,.\, t$$

one-argument function

Key Idea

λ-abstraction (or anonymous function):

$$\lambda x.\,t$$

one-argument function

formal argument

Key Idea

λ-abstraction (or anonymous function):

$$\lambda x.t$$

one-argument function

function body

formal argument

Syntax

t	$\longrightarrow$		terms:
		x	variable
	$\mid$	$\lambda x.t$	abstraction
	$\mid$	$t\ t$	application

Syntax

$$
\begin{array}{rcll}
t & \rightarrow & & \text{terms:} \\
 & & x & \text{variable} \\
 & | & \lambda x.t & \text{abstraction} \\
 & | & t\ t & \text{application}
\end{array}
$$

Note:

Syntax

$$
\begin{array}{rcll}
t & \rightarrow & & \text{terms:} \\
 & & x & \text{variable} \\
 & | & \lambda x.t & \text{abstraction} \\
 & | & t\ t & \text{application}
\end{array}
$$

Note:

- x is the syntactic category of variables. We will use actual names like x, y, z, u, v, w, $\ldots$

Syntax

$$
\begin{array}{rcll}
t & \rightarrow & & \text{terms:} \\
& & x & \text{variable} \\
& | & \lambda x.t & \text{abstraction} \\
& | & t\ t & \text{application}
\end{array}
$$

Note:

- x is the syntactic category of variables. We will use actual names like x, y, z, u, v, w, ...
- λ-abstractions often named for convenience. E.g. $I \equiv \lambda x.x$.

Syntax

$$
\begin{array}{rcll}
t & \rightarrow & & \text{terms:} \\
 & & x & \text{variable} \\
 & | & \lambda x.t & \text{abstraction} \\
 & | & t\ t & \text{application}
\end{array}
$$

Note:

- x is the syntactic category of variables. We will use actual names like x, y, z, u, v, w, . . .
- λ-abstractions often named for convenience. E.g. $I \equiv \lambda x.x$. ***Just an abbreviation!***

Syntax

$$\begin{array}{rcll} t & \to & & \text{terms:} \\ & & x & \text{variable} \\ & | & \lambda x.t & \text{abstraction} \\ & | & t\ t & \text{application} \end{array}$$

Note:

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

Scope

Scope

- An ***occurrence*** of x is ***bound*** if it occurs in the body t of a λ-abstraction $\lambda x.t$.

Scope

- An ***occurrence*** of x is ***bound*** if it occurs in the body t of a λ-abstraction $\lambda x.t$.
- A non-bound occurrence is ***free***.

Scope

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

Scope

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

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

Operational Semantics (1)

Sole means of computation: ***β-reduction*** or ***function application***:

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

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.

Operational Semantics (2)

A term that can be β-reduced is called a ***(β-)redex***.

Exercise: Underline the redexes in

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