

Functional Programming

WS 2007/08

Christian Sternagel¹ (VO + PS)
Friedrich Neurauter² (PS)
Harald Zankl³ (PS)

Computational Logic
Institute of Computer Science
University of Innsbruck

16 November 2007

¹christian.sternagel@uibk.ac.at

²friedrich.neurauter@uibk.ac.at

³harald.zankl@uibk.ac.at

Overview

Week 5 - λ -Calculus

Summary of Week 4

λ -Calculus - Introduction

λ -Calculus - Formalities

λ -Calculus - Data Types

Overview

Week 5 - λ -Calculus

Summary of Week 4

λ -Calculus - Introduction

λ -Calculus - Formalities

λ -Calculus - Data Types

Binary Trees

- ▶ at most 2 children per node
- ▶ used for searching
- ▶ Huffman coding

Huffman Coding

Idea

- ▶ use shortest codewords for most frequent symbols

Usage

- ▶ compression

Raison D'être

Goal

- ▶ find a framework in which **every** algorithm can be defined
- ▶ universal language

Result

- ▶ Turing machines
- ▶ λ -Calculus
- ▶ ...

Overview

Week 5 - λ -Calculus

Summary of Week 4

λ -Calculus - Introduction

λ -Calculus - Formalities

λ -Calculus - Data Types

Syntax

λ -Terms

$$t ::= \overbrace{x}^{\text{Variable}} \mid \underbrace{(\lambda x.t)}_{\text{Abstraction}} \mid \overbrace{(t t)}^{\text{Application}}$$

$\mathcal{T}(\mathcal{V})$ set of **all** λ -terms over set of variables \mathcal{V}

Conventions (omit outermost parentheses)(combine nested lambdas)(application is left-associative)

$$\begin{aligned} &\lambda x.x \\ &\lambda xy.x \\ &\lambda xyz.x z (y z) \end{aligned}$$

Intuition

Example (λ -terms)

- ▶ $\lambda x. \text{add } x \ \bar{1}$
- ▶ $(\lambda x. \text{add } x \ \bar{1}) \ \bar{2}$
- ▶ `if true $\bar{1}$ $\bar{0}$`
- ▶ `pair $\bar{2}$ $\bar{4}$`
- ▶ `fst (pair $\bar{2}$ $\bar{4}$)`

Example (OCaml equivalent)

- ▶ `(fun x -> x + 1)`
- ▶ `(fun x -> x + 1) 2` $\rightarrow^+ 3$
- ▶ `if true then 1 else 0` $\rightarrow 1$
- ▶ `(2, 4)`
- ▶ `fst (2, 4)` $\rightarrow 2$

Remark

' $\bar{0}$ ', ' $\bar{1}$ ', ' $\bar{2}$ ', ' $\bar{3}$ ', ' $\bar{4}$ ', 'add', 'fst', 'if', 'pair', and 'true' are just abbreviations for more complex λ -terms

Computations

Idea

- ▶ rules to manipulate λ -terms
- ▶ a single rule is enough

The β -rule

$$(\lambda x. s) t \rightarrow_{\beta} s\{x \mapsto t\}$$

substitute x by t in s

- ▶ application of a function to some input

Examples

$$\begin{aligned}
 (\lambda x. x) (\lambda x. x) &\rightarrow_{\beta} \lambda x. x \\
 (\lambda xy. y) (\lambda x. x) &\rightarrow_{\beta} \lambda y. y \\
 (\lambda xyz. x z (y z)) (\lambda x. x) &\rightarrow_{\beta} \lambda yz. (\lambda x. x) z (y z) \\
 (\lambda x. x x) (\lambda x. x x) &\rightarrow_{\beta} (\lambda x. x x) (\lambda x. x x) \\
 \lambda x. x &\rightarrow_{\beta} \text{no } \beta\text{-step possible}
 \end{aligned}$$

Overview

Week 5 - λ -Calculus

Summary of Week 4

λ -Calculus - Introduction

λ -Calculus - Formalities

λ -Calculus - Data Types

Subterms

Definition

$\text{Sub}(t)$ is set of subterms of t

$$\text{Sub}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{t\} \cup \text{Sub}(u) & t = \lambda x.u \\ \{t\} \cup \text{Sub}(u) \cup \text{Sub}(v) & t = u v \end{cases}$$

Example

$$\begin{aligned} \text{Sub}(\lambda xy.x) &= \{\lambda xy.x\} \cup \text{Sub}(\lambda y.x) \\ &= \{\lambda xy.x, \lambda y.x\} \cup \text{Sub}(x) \\ &= \{\lambda xy.x, \lambda y.x, x\} \end{aligned}$$

Examples

t	$\mathcal{V}\text{ar}(t)$	$\mathcal{F}\mathcal{V}\text{ar}(t)$	$\mathcal{B}\mathcal{V}\text{ar}(t)$
$\lambda x.x$	$\{x\}$	\emptyset	$\{x\}$
$x y$	$\{x, y\}$	$\{x, y\}$	\emptyset
$(\lambda x.x) x$	$\{x\}$	$\{x\}$	$\{x\}$
$\lambda x.x y z$	$\{x, y, z\}$	$\{y, z\}$	$\{x\}$

Free and Bound Variables

Definition

$$\text{variables } \mathcal{V}\text{ar}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{x\} \cup \mathcal{V}\text{ar}(u) & t = \lambda x.u \\ \mathcal{V}\text{ar}(u) \cup \mathcal{V}\text{ar}(v) & t = u v \end{cases}$$

free variables

$$\mathcal{F}\mathcal{V}\text{ar}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \mathcal{F}\mathcal{V}\text{ar}(u) \setminus \{x\} & t = \lambda x.u \\ \mathcal{F}\mathcal{V}\text{ar}(u) \cup \mathcal{F}\mathcal{V}\text{ar}(v) & t = u v \end{cases}$$

bound variables

$$\mathcal{B}\mathcal{V}\text{ar}(t) \stackrel{\text{def}}{=} \begin{cases} \emptyset & t = x \\ \{x\} \cup \mathcal{B}\mathcal{V}\text{ar}(u) & t = \lambda x.u \\ \mathcal{B}\mathcal{V}\text{ar}(u) \cup \mathcal{B}\mathcal{V}\text{ar}(v) & t = u v \end{cases}$$

Substitutions

Definition

function from variables to terms

$$\sigma: \mathcal{V} \rightarrow \mathcal{T}(\mathcal{V})$$

such that only for finitely many $x \in \mathcal{V}$, $\sigma(x) \neq x$

Notation

set of bindings for all $x \in \mathcal{V}$ with $\sigma(x) \neq x$

$$\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$$

Example

$$\sigma = \{x \mapsto z, y \mapsto \lambda x.x\} \text{ hence } \sigma(x) = z \text{ and } \sigma(y) = \lambda x.x$$

Substitutions (cont'd)

Definition (Domain)

domain of σ (i.e., altered variables)

$$\text{Dom}(\sigma) \stackrel{\text{def}}{=} \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$$

Definition (Restriction)

restriction of σ to set of variables X

$$\sigma|_X \stackrel{\text{def}}{=} \{x \mapsto t \in \sigma \mid x \in X\}$$

Example

$$\begin{aligned} \text{Dom}(\{x \mapsto z, y \mapsto \lambda x.x\}) &= \{x, y\} \\ \{x \mapsto z, y \mapsto \lambda x.x\}|_{\{x\}} &= \{x \mapsto z\} \end{aligned}$$

Substitutions (cont'd)

Definition (Application)

apply substitution σ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} \sigma(t) & t = x \\ \lambda x.(u\sigma|_{\mathcal{F}\text{Var}(t)}) & t = \lambda x.u \\ (u\sigma) (v\sigma) & t = u v \end{cases}$$

Example

$$\sigma = \{x \mapsto \lambda x.x, z \mapsto \lambda x.x x\}$$

$$\begin{aligned} x\sigma &= \lambda x.x \\ y\sigma &= y \\ (\lambda x.x)\sigma &= \lambda x.x \\ (\lambda x.x z)\sigma &= \lambda x.x (\lambda x.x x) \\ (z z)\sigma &= (\lambda x.x x) (\lambda x.x x) \end{aligned}$$

 β -Reduction

Definition (Context)

context $C \in \mathcal{C}(\mathcal{V})$

$$C ::= \square \mid \lambda x.C \mid C t \mid t C$$

with $x \in \mathcal{V}$ and $t \in \mathcal{T}(\mathcal{V})$

- ▶ $C[s]$ denotes replacing \square by term s in context C

Example

$$\begin{aligned} C_1 &= \square \\ C_2 &= x \square \\ C_3 &= \lambda x.\square x \\ C_1[\lambda x.x] &= \lambda x.x \\ C_2[\lambda x.x] &= x (\lambda x.x) \\ C_3[\lambda x.x] &= \lambda x.(\lambda x.x) x \end{aligned}$$

 β -Reduction (cont'd)Definition (β -step)

if exist context C and terms s , u , and v such that

$$s = C[(\lambda x.u) v]$$

then

$$s \rightarrow_{\beta} C[u\{x \mapsto v\}]$$

is a **β -step** with **redex** $(\lambda x.u) v$ and **contractum** $u\{x \mapsto v\}$

- ▶ $s \rightarrow_{\beta}^+ t$ denotes sequence $s = t_1 \rightarrow_{\beta} t_2 \rightarrow_{\beta} \dots \rightarrow_{\beta} t_n = t$ with $n > 0$
- ▶ $s \rightarrow_{\beta}^* t$ is sequence with $n \geq 0$ (s **β -reduces** to t)

Example

$$\begin{aligned} \Omega &= (\lambda x.x x) (\lambda x.x x) & K_* \Omega &\rightarrow_{\beta} K_* \Omega \rightarrow_{\beta} \dots \\ K_* &= \lambda xy.y & K_* \Omega &\rightarrow_{\beta} \lambda y.y \end{aligned}$$

Problem

Example

- ▶ consider $\lambda xy.x$
- ▶ behavior: “take 2 arguments, ignore second, return first”
- ▶ $(\lambda xy.x) y z \rightarrow_{\beta} (\lambda y.y) z \rightarrow_{\beta} z$
- ▶ clearly not intended (Problem: **variable capture**)

Idea

- ▶ always rename bound variables before applying substitution
- ▶ $(\lambda xy.x) y z \rightarrow_{\alpha} (\lambda x'y'.x') y z \rightarrow_{\beta} (\lambda y'.y) z \rightarrow_{\beta} y$

What Are the Results of Computations?

Idea

- ▶ only **terms** in λ -calculus
- ▶ express functions **and** values through λ -terms

Definition (Normal form)

$t \in \mathcal{T}(\mathcal{V})$ is in **normal form** if no β -step is applicable

Example

$\lambda x.x$ NF
 $(\lambda x.x) y$ not in NF

Solution

The α -rule

$$C[\lambda x.t] \rightarrow_{\alpha} C[\lambda x'.t\{x \mapsto x'\}] \quad \text{if } x' \notin \text{Var}(t)$$

- ▶ always possible in both directions, i.e., $s \rightarrow_{\alpha} t \iff t \rightarrow_{\alpha} s$
- ▶ hence **α -conversion** (\leftrightarrow_{α})

Avoid variable capture

- ▶ always α -convert before applying β

Overview

Week 5 - λ -Calculus

Summary of Week 4

λ -Calculus - Introduction

λ -Calculus - Formalities

λ -Calculus - Data Types

Booleans and Conditionals

OCaml

- ▶ **true**
- ▶ **false**
- ▶ **if** b **then** t **else** e

 λ -Calculus

- ▶ $\text{true} \stackrel{\text{def}}{=} \lambda xy.x$
- ▶ $\text{false} \stackrel{\text{def}}{=} \lambda xy.y$
- ▶ $\text{if} \stackrel{\text{def}}{=} \lambda xyz.x\ y\ z$

Example

if true $x\ y \rightarrow_{\beta} \text{true}\ x\ y \rightarrow_{\beta} x$
 if false $x\ y \rightarrow_{\beta} \text{false}\ x\ y \rightarrow_{\beta} y$

Natural Numbers

Definition

$$s^0\ t \stackrel{\text{def}}{=} t$$

$$s^{n+1}\ t \stackrel{\text{def}}{=} s\ (s^n\ t)$$

OCaml

- ▶ 0
- ▶ 1
- ▶ n
- ▶ (+)
- ▶ (*)
- ▶ (**)

 λ -Calculus

- ▶ $\bar{0} \stackrel{\text{def}}{=} \lambda fx.x$
- ▶ $\bar{1} \stackrel{\text{def}}{=} \lambda fx.f\ x$
- ▶ $\bar{n} \stackrel{\text{def}}{=} \lambda fx.f^n\ x$
- ▶ $\text{add} \stackrel{\text{def}}{=} \lambda mnfx.m\ f\ (n\ f\ x)$
- ▶ $\text{mul} \stackrel{\text{def}}{=} \lambda mnf.m\ (n\ f)$
- ▶ $\text{exp} \stackrel{\text{def}}{=} \lambda mn.n\ m$

Pairs

OCaml

- ▶ **fun** $x\ y \rightarrow (x, y)$
- ▶ **fst**
- ▶ **snd**

 λ -Calculus

- ▶ $\text{pair} \stackrel{\text{def}}{=} \lambda xyf.f\ x\ y$
- ▶ $\text{fst} \stackrel{\text{def}}{=} \lambda p.p\ \text{true}$
- ▶ $\text{snd} \stackrel{\text{def}}{=} \lambda p.p\ \text{false}$

Lists

OCaml

- ▶ ::
- ▶ hd
- ▶ tl
- ▶ []
- ▶ **fun** $x \rightarrow x = []$

 λ -Calculus

- ▶ $\text{cons} \stackrel{\text{def}}{=} \lambda xy.\text{pair}\ \text{false}\ (\text{pair}\ x\ y)$
- ▶ $\text{hd} \stackrel{\text{def}}{=} \lambda z.\text{fst}\ (\text{snd}\ z)$
- ▶ $\text{tl} \stackrel{\text{def}}{=} \lambda z.\text{snd}\ (\text{snd}\ z)$
- ▶ $\text{nil} \stackrel{\text{def}}{=} \lambda x.x$
- ▶ $\text{null} \stackrel{\text{def}}{=} \text{fst}$

Recursion

OCaml

```
let rec length x = if x = [] then 0 else 1 + length (tl x);;
```

λ -Calculus

$$\text{length} \stackrel{\text{def}}{=} \mathbf{Y} (\lambda f x. \text{if } (\text{null } x) \bar{0} (\text{add } \bar{1} (f (\text{tl } x))))$$

Definition (Y-combinator)

$$\mathbf{Y} \stackrel{\text{def}}{=} \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))$$

\mathbf{Y} has **fixed point property**, i.e., for all $t \in \mathcal{T}(\mathcal{V})$

$$\mathbf{Y} t \rightarrow_{\beta}^* t (\mathbf{Y} t)$$