

Functional Programming

WS 2009/10

Christian Sternagel (VO)
Friedrich Neurauter (PS) Sarah Winkler (PS)

Computational Logic
Institute of Computer Science
University of Innsbruck

6 November 2009

Week 5 - λ -Calculus

Summary of Week 4

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

This Week

Practice I

OCaml introduction, lists, strings, trees

Theory I

lambda-calculus, evaluation strategies, induction, reasoning about functional programs

Practice II

efficiency, tail-recursion, combinator-parsing

Theory II

type checking, type inference

Advanced Topics

lazy evaluation, infinite data structures, monads, ...

Origin

Goal

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

Result

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

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)

$$\lambda x.x$$

$$\lambda xy.x$$

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

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

Intuition

Example

λ -terms

- ▶ $\lambda x. \text{ADD } x \ 1$
- ▶ $(\lambda x. \text{ADD } x \ 1) \ 2$
- ▶ $\text{IF TRUE } 1 \ 0$
- ▶ $\text{PAIR } 2 \ 4$
- ▶ $\text{FST}(\text{PAIR } 2 \ 4)$

OCaml

- ▶ `(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

'0', '1', '2', '3', '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

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

substitute x by t in s

application of a function to some input

Examples

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

Subterms

Definition

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

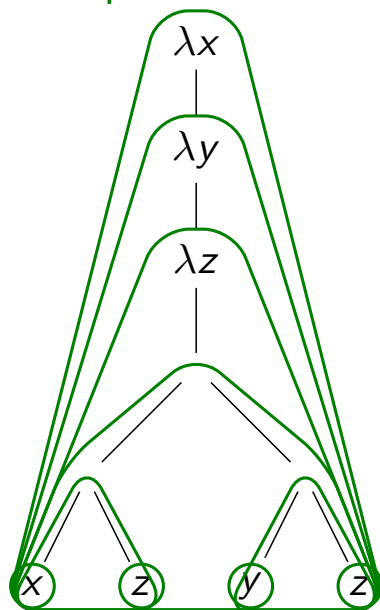
$$\text{Sub}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{t\} \cup \text{Sub}(u) & t = \lambda x.u \\ \{t\} \cup \text{Sub}(u) \cup \text{Sub}(v) & t = u v \end{cases}$$

Example

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

Syntax Trees

Example



$$t = \lambda xyz.x z (y z)$$

$$\text{Sub}(t) = \{t, \lambda yz.x z (y z), \lambda z.x z (y z), x z (y z), x z, y z, x, z, y\}$$

Variables

Definition 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 and Bound Variables

Definition

free variables

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

bound variables

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

Examples

t	$\mathcal{Var}(t)$	$\mathcal{FVar}(t)$	$\mathcal{BVar}(t)$
$\lambda x.x$	$\{x\}$	\emptyset	$\{x\}$
$x y$	$\{x, y\}$	$\{x, y\}$	\emptyset
$(\lambda x.x) x$	$\{x\}$	$\{x\}$	$\{x\}$
$\lambda x.x y z$	$\{x, y, z\}$	$\{y, z\}$	$\{x\}$

Substitutions

Definition

function from variables to terms

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

in our case we only need substitutions replacing a single variable, i.e., only for one $x \in \mathcal{V}$, $\sigma(x) \neq x$

Notation

binding for x such that $\sigma(x) \neq x$

$$\sigma = \{x/t\}$$

Example

$$\sigma = \{x/\lambda x.x\} \text{ hence } \sigma(x) = \lambda x.x \text{ and } \sigma(y) = y$$

Substitutions (cont'd)

Definition (Application)

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y \neq x \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x.u & t = \lambda x.u \\ \lambda y.(u\sigma) & t = \lambda y.u \text{ and } x \neq y \end{cases}$$

Example

$$\sigma = \{x/\lambda x.x\}$$

$$x\sigma = \lambda x.x$$

$$y\sigma = y$$

$$(\lambda x.x)\sigma = \lambda x.x$$

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

Contexts

Example

$$C_1 = \square$$

$$C_2 = x \square$$

$$C_3 = \lambda x. \square x$$

$$C_1[\lambda x.x] = \lambda x.x$$

$$C_2[\lambda x.x] = x (\lambda x.x)$$

$$C_3[\lambda x.x] = \lambda x. (\lambda x.x) x$$

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

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

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

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

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$K_* \Omega \rightarrow_{\beta} K_* \Omega \rightarrow_{\beta} \cdots$$

$$K_* \Omega \rightarrow_{\beta} \lambda y.y$$

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

rename bound variables where necessary

Solution

Definition (Application (recall))

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y \neq x \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x.u & t = \lambda x.u \\ \lambda y.(u\sigma) & t = \lambda y.u \text{ and } x \neq y \end{cases}$$

Solution

Definition (Application (revised))

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y \neq x \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x.u & t = \lambda x.u \\ \lambda y.(u\sigma) & t = \lambda y.u \text{ and } x \neq y \text{ and } y \notin \mathcal{FVar}(s) \\ \lambda z.(u\{y/z\}\sigma) & t = \lambda y.u \text{ and } x \neq y \text{ and } y \in \mathcal{FVar}(s) \end{cases}$$

where z is assumed to be **fresh**

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

$$\begin{array}{ll} \lambda x.x & \text{NF} \\ (\lambda x.x) y & \text{not in NF} \end{array}$$

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

$$\begin{aligned} \text{IF TRUE } x y &\rightarrow_{\beta}^+ \text{TRUE } x y \rightarrow_{\beta}^+ x \\ \text{IF FALSE } x y &\rightarrow_{\beta}^+ \text{FALSE } x y \rightarrow_{\beta}^+ y \end{aligned}$$

Natural Numbers

Definition

$$\begin{aligned} s^0 t &\stackrel{\text{def}}{=} t \\ s^{n+1} t &\stackrel{\text{def}}{=} s (s^n t) \end{aligned}$$

OCaml vs. λ -Calculus

$$\begin{aligned} 0 & \stackrel{\text{def}}{=} \lambda fx.x \\ 1 & \stackrel{\text{def}}{=} \lambda fx.f x \\ n & \stackrel{\text{def}}{=} \lambda fx.f^n x \\ (+) & \text{ADD} \stackrel{\text{def}}{=} \lambda mnfx.m f (n f x) \\ (*) & \text{MUL} \stackrel{\text{def}}{=} \lambda mnf.m (n f) \\ (**) & \text{EXP} \stackrel{\text{def}}{=} \lambda mn.n m \end{aligned}$$

Pairs

OCaml vs. λ -Calculus

<code>fun x y -> (x,y)</code>	$\text{PAIR} \stackrel{\text{def}}{=} \lambda xyf.f x y$
<code>fst</code>	$\text{FST} \stackrel{\text{def}}{=} \lambda p.p \text{ TRUE}$
<code>snd</code>	$\text{SND} \stackrel{\text{def}}{=} \lambda p.p \text{ FALSE}$

Lists

OCaml vs. λ -Calculus

<code>::</code>	$\text{CONS} \stackrel{\text{def}}{=} \lambda xy.\text{PAIR FALSE (PAIR } x y)$
<code>hd</code>	$\text{HD} \stackrel{\text{def}}{=} \lambda z.\text{FST (SND } z)$
<code>tl</code>	$\text{TL} \stackrel{\text{def}}{=} \lambda z.\text{SND (SND } z)$
<code>[]</code>	$\text{NIL} \stackrel{\text{def}}{=} \lambda x.x$
<code>fun x -> x = []</code>	$\text{NULL} \stackrel{\text{def}}{=} \text{FST}$

Recursion

OCaml

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

λ -Calculus

$$\text{LENGTH} \stackrel{\text{def}}{=} Y (\lambda f x. \text{IF} (\text{NULL } x) 0 (\text{ADD } 1 (f (\text{TL } x))))$$

Definition (Y-combinator)

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

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

$$Y t \leftrightarrow^* t (Y t)$$