Functional Programming
WS 2007/08

Christian Sternagel[1] (VO + PS)
Friedrich Neurauter[2] (PS)
Harald Zankl[3] (PS)

Computational Logic
Institute of Computer Science

University of Innsbruck

16 November 2007

[1] christian.sternagel@uibk.ac.at
[2] friedrich.neurauter@uibk.ac.at
[3] harald.zankl@uibk.ac.at

Overview

Week 5 - λ-Calculus

- Summary of Week 4
- λ-Calculus - Introduction
- λ-Calculus - Formalities
- λ-Calculus - Data Types

Overview

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

Overview

Raison D'être

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)

$$\lambda x.x$$

$$\lambda xy.x$$

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

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

$$(\lambda x.s)\ t \rightarrow_\beta \underbrace{s\{x \mapsto t\}}_{\text{substitute } x \text{ by } t \text{ in } s}$$

- application of a function to some input

Examples

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

Overview

Subterms

Definition

$\mathcal{S}\text{ub}(t)$ is set of subterms of t

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

Example

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

Free and Bound 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 variables

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

bound variables

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

Examples

t	$\mathcal{V}\text{ar}(t)$	$\mathcal{FV}\text{ar}(t)$	$\mathcal{BV}\text{ar}(t)$
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\varnothing$
$(\lambda x.x)\ x$	$\{x\}$	$\{x\}$	$\{x\}$
$\lambda x.x\ y\ z$	$\{x, y, z\}$	$\{y, z\}$	$\{x\}$

Substitutions

Definition

function from variables to terms

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

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 $\sigma(x) \neq x$

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

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)

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

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

$$\mathcal{D}\mathrm{om}(\{x \mapsto z, y \mapsto \lambda x.x\}) = \{x, y\}$$
$$\{x \mapsto z, y \mapsto \lambda x.x\}|_{\{x\}} = \{x \mapsto z\}$$

Substitutions (cont'd)

Definition (Application)

apply substitution σ to term t

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

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

$$C_1 = \square$$
$$C_2 = x\ \square$$
$$C_3 = \lambda x.\square\ x$$
$$C_1[\lambda x.x] = \lambda x.x$$
$$C_2[\lambda x.x] = x\ (\lambda x.x)$$
$$C_3[\lambda x.x] = \lambda x.(\lambda x.x)\ x$$

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

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

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

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

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

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

Solution

The α-rule

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

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

Avoid variable capture

- always α-convert before applying β

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

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

Overview

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

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

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 −> (x, y)
- fst
- snd

λ-Calculus

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

Lists

OCaml

- ::
- hd
- tl
- []
- **fun** x −> x = []

λ-Calculus

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

Recursion

OCaml

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

λ-Calculus

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

Definition (Y-combinator)

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

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

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