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

Huffman Coding

Idea

- use shortest codewords for most frequent symbols

Usage

- compression

Binary Trees

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

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	OCaml
▶ λx.ADD x 1	▶ `(fun x -> x+1)`
▶ (λx.ADD x 1) 2	▶ `(fun x -> x+1) 2` $\rightarrow^+$ `3`
▶ IF TRUE 1 0	▶ `if true then 1 else 0` $\rightarrow$ `1`
▶ PAIR 2 4	▶ `(2,4)`
▶ FST(PAIR 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

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

application of a function to some input

Examples

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

Subterms

Definition

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

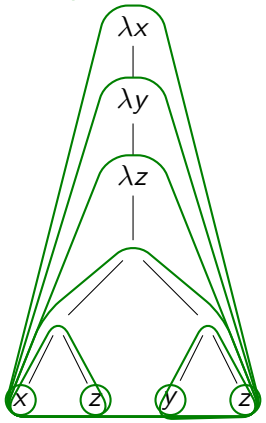
$$\mathcal{S}\text{ub}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{t\} \cup \mathcal{S}\text{ub}(u) & t = \lambda x.u \\ \{t\} \cup \mathcal{S}\text{ub}(u) \cup \mathcal{S}\text{ub}(v) & t = u\ v \end{cases}$$

Example

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

Syntax Trees

Example



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

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

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

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

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

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

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

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

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

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*"
- $(\lambda xy.x)\ y\ z \to_\beta (\lambda y.y)\ z \to_\beta z$
- 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

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

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

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

Booleans and Conditionals

OCaml

- `true`
- `false`
- `if` b `then` t `else` e

λ-Calculus

- TRUE $\stackrel{\text{def}}{=} \lambda xy.x$
- FALSE $\stackrel{\text{def}}{=} \lambda xy.y$
- IF $\stackrel{\text{def}}{=} \lambda xyz.x\ y\ z$

Example

$$\text{IF TRUE } x\ y \to_\beta^+ \text{TRUE } x\ y \to_\beta^+ x$$
$$\text{IF FALSE } x\ y \to_\beta^+ \text{FALSE } x\ y \to_\beta^+ y$$

Natural Numbers

Definition

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

OCaml vs. λ-Calculus

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

Pairs

OCaml vs. λ-Calculus

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

Lists

OCaml vs. λ-Calculus

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

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

Definition (Y-combinator)

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

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

$$\text{Y}\ t \leftrightarrow^{*} t\ (\text{Y}\ t)$$