Functional Programming

WS 2013/14

Harald Zankl (VO+PS)
Cezary Kaliszyk (PS)

Computational Logic
Institute of Computer Science
University of Innsbruck

week 5

Summary of Week 4

Binary Trees

- at most 2 children per node
- applications
  - search trees
  - Huffman coding

Huffman Coding

- Idea: use shortest codewords for most frequent symbols
- Application: (lossless) data 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 dynamic programming

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 (Turing, 1930s)
- λ-Calculus (Church, 1930s)
- . . .

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

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

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)

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

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 (combine nested lambdas)

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

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 (application is left-associative and binds strongest)

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

Intuition

Example

λ-terms	OCaml
▶ $\lambda x.\text{add } x\ \overline{1}$	▶ `fun x -> x+1`
▶ $(\lambda x.\text{add } x\ \overline{1})\ \overline{2}$	▶ `(fun x -> x+1) 2` $\to^+$ `3`
▶ $\text{if true } \overline{1}\ \overline{0}$	▶ `if true then 1 else 0` $\to$ `1`
▶ $\text{pair } \overline{2}\ \overline{4}$	▶ `(2,4)`
▶ $\text{fst}(\text{pair } \overline{2}\ \overline{4})$	▶ `fst(2,4)` $\to$ `2`
▶ $\lambda xy.\text{add } x\ y$	▶ `fun x y -> x + y`
▶ $\lambda x.(\lambda y.\text{add } x\ y)$	▶ `fun x -> fun y -> x + y`

Remark

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

Subterms

Definition

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

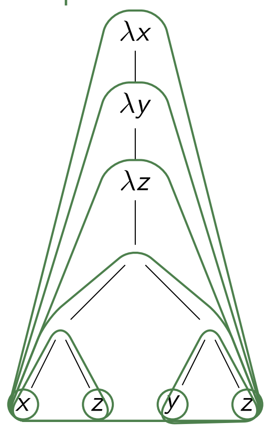
$$\mathcal{S}\text{ub}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{t\} \cup \mathcal{S}\text{ub}(u) & t = \lambda x.u \\ \{t\} \cup \mathcal{S}\text{ub}(u) \cup \mathcal{S}\text{ub}(v) & t = u\ v \end{cases}$$

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

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

A λ-term without free variables is called closed.

Examples

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

Computations

Idea

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

The β-rule (informal)

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

application of a function to some input

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (`fun x y -> x` in OCaml)
- behavior: "*take 2 arguments, ignore second, return first*"
- $(\lambda xy.x)\ v\ w \rightsquigarrow (\lambda y.v)\ w \rightsquigarrow v$ ✓
- $(\lambda xy.x)\ y\ z \rightsquigarrow (\lambda y.y)\ z \rightsquigarrow z$ ✗
- clearly not intended (Problem: variable capture)
- $(\lambda xy.x)\ y\ z \to_\beta (\lambda y'.y)\ z \to_\beta y$

Solution

rename bound variables where necessary

Ocaml

```
let y = 3 and z = 2;;
(fun u -> (fun v -> u)) y z;;
```

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

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

Substitutions (cont'd)

Definition (Application)

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

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y,\ x \neq y \\ (u\sigma)\ (v\sigma) & t = u\ v \\ \lambda x.u & t = \lambda x.u \\ \lambda y.(u\sigma) & t = \lambda y.u,\ x \neq y,\ y \notin \mathcal{FV}\mathrm{ar}(s) \\ \lambda y'.((u\{y/y'\})\sigma) & t = \lambda y.u,\ x \neq y,\ y \in \mathcal{FV}\mathrm{ar}(s),\ y' \text{ fresh} \end{cases}$$

Example ($\sigma = \{x/\lambda v.v\ w\}$)

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

Examples

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

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

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

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

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

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

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

β-Reduction

Example

$$\Omega = (\lambda x.x\ x)\ (\lambda x.x\ x)$$
$$K_* = \lambda xy.y$$
$$I_2 = \lambda xy.x\ y$$

$K_*\ \Omega \to_\beta K_*\ \Omega \to_\beta \cdots$

$K_*\ \Omega \to_\beta \lambda y.y$

$I_2\ I_2 = (\lambda xy.x\ y)\ (\lambda xy.x\ y) \to_\beta \lambda y.(\lambda xy.x\ y)\ y \equiv \lambda y.(\lambda xy'.x\ y')\ y$

$\to_\beta \lambda y.(\lambda y'.y\ y') = \lambda yy'.y\ y' \equiv I_2$

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 (NF) if no β-step possible

Example

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

Lambda Interpreter for Pure Students

developed by Michael Brunner (bachelor thesis)

λ-Terms

$$t ::= \texttt{x} \mid (\backslash\texttt{x}.t) \mid (t\ t)$$

Conventions

- interpreter command `!pretty` toggles use of conventions for printing
- nested abstractions use spaces to separate variable names, e.g.,

$\lambda xy.x$	`\x y.x`
$\lambda x_1.y$	`\x1.y`

Result

Normal Forms

- result of input is corresponding NF
  - ```
    > (\x.x) (\x.x)
    NF: (\x.x)
    ```

Evaluation Strategy

- `!by_value` activates call-by-value evaluation (next lecture)
- `!by_name` activates call-by-name evaluation (next lecture)
- `!trace` toggles tracing

Abbreviations & Initialisation

Interpreter Command

$$\texttt{!def}\ \langle \mathit{name} \rangle = t$$

Example

```
> !def I = \x.x
> !def K = \x y.x
> !def S = \x y z.x z (y z)
> S K I
NF: \z.z
```

.lambdainit

content of file `.lambdainit` is loaded on start-up of `lips`
```