CS101C

Type Theory and Formal Methods

Lecture 5

April 14, 2003

λ-Calculus

Invented in 1932-33 by Church.

Idea: standard notation for a function taking an argument: $\lambda x.e$

OCaml: `fun x -> e`

SML: `fn x => e`

Lisp: `(lambda (x)(e))`

Haskell: `\x -> e`

λ-Calculus

- Variables: a single variable is a λ-term.
- Functions: $\lambda x.t$, where t is an arbitrary λ-term. (Or using a "smart" syntax: $\lambda x.t[x]$, where t is a *meta-variable*.)
- Application: $t_1(t_2)$ (or just: $t_1\ t_2$), where t_1 and t_2 are arbitrary λ-terms.
- β-reduction:

$$\lambda x.t\ t_2 \leftrightarrow t \text{ with } t_2 \text{ substituted for } x$$

  or in "smart" syntax:

$$\lambda x.t[x]\ t_2 \leftrightarrow t[t_2]$$

Examples

- $(\lambda x.x)\,y \;\rightarrow\; y$
- $(\lambda x.y)\,z \;\rightarrow\; y$
- $(\lambda f.f\ x)\,(\lambda y.y) \;\rightarrow\; (\lambda y.y)\,x$
- $(\lambda f.f\ f)\,(\lambda f.f\ f) \;\rightarrow\; (\lambda f.f\ f)\,(\lambda f.f\ f)$

Free variables

- Variable x is free is a λ-term x.
- If t is a λ-term, then all free variables of t, are free in $\lambda x.t$, except for x. All *free* occurrences of x in t become *bound* in $\lambda x.t$ and the $\lambda x.$ is a binding occurrence for them.
- If t_1, t_2 are λ-terms, then all free occurrences of t_1 and t_2 are also free in $t_1\ t_2$.

Examples:

- $\lambda x.x$
- $x\ \lambda x.x$
- $\lambda x.(x\ \lambda x.x)$

α-equality

Names of bound variables do not matter!

If some binding occurrences are renamed together with the corresponding bound occurrences and the *binding structure* remains the same (e.g. each bound position renames bound by the *same* λ), then the resulting term is *α-equal* to the original one.

Example: $\lambda x.x =_\alpha \lambda y.y$

Note: $\lambda x.\lambda y.x \neq_\alpha \lambda y.\lambda y.y$ — here a *capture* happened.

Capture-avoiding substitution

Suppose t and t' are λ-terms. To perform *capture-avoiding substitution* of t' for x in t

- α-rename *bound* variables in t to avoid collisions with *free* variables in t'
- replace all *free* occurrences of x in t with t'

Examples:

- $\lambda x.x$ with y substituted for x is $\lambda x.x$
- $\lambda y.x$ with y substituted for x is $\lambda z.y$
- $x\ (\lambda x.x)$ with $\lambda x.x$ substituted for x is $(\lambda x.x)\ (\lambda x.x)$

λ-Calculus

- Variables: a single variable is a λ-term.
- Functions: $\lambda x.t$, where t is an arbitrary λ-term. (Or using a "smart" syntax: $\lambda x.t[x]$, where t is a *meta-variable*.)
- Application: $t_1(t_2)$ (or just: $t_1\ t_2$), where t_1 and t_2 are arbitrary λ-terms.
- β-reduction:

$$\lambda x.t\ t_2 \leftrightarrow t \text{ with } t_2 \text{ substituted for } x$$

or in "smart" syntax:

$$\lambda x.t[x]\ t_2 \leftrightarrow t[t_2]$$

Meta-Variables as Patterns

$t[]$ matches any λ-term

$\lambda x.t[x]$ matches any λ-term with λ on top

Semantics: $t[\bullet]$ can not have *free* occurrences of x

$\lambda x.t[]$ matches any λ-term with λ on top, where the body does not have any variable occurrences bound by that top λ

$t[x]$ matches any λ-term

Semantics: $t[\bullet]$ *can* have free occurrences of x

Examples

λ-term	Pattern to match		
	$\lambda x.t[x]$	$\lambda y.t[]$	$\lambda x.\lambda y.t[x]$
$\lambda z.z$	**Y**	**N**	**N**
$\lambda z.x$	**Y**	**Y**	**N**
$\lambda y.\lambda y.y$	**Y**	**Y**	**N**
$\lambda x.\lambda y.(x\ \lambda y.y)$	**Y**	**N**	**Y**