Functional Programming and Scheme

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

The λ-Calculus

A pure λ-term t is defined inductively as follows:

- Any variable x is a λ-term
- If t is a λ-term, so is $\lambda x.\, t$ (abstraction)
- If t_1, t_2 are λ-terms, so is $t_1 t_2$ (application)

We use $x, y, z, \ldots$ for variables

The definition above defines an infinite set, named t

α-Reduction

Replacing all bound variables gives the same term

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

Analogy in C:

```
int f (int x) {return x+1}
```

Is same as

```
int f (int y) {return y+1}
```

λ-Calculus Evaluation (Informal)

Identity function: $\lambda x.\, x$

$(\lambda x.\, x)\; y = y$

Observation: we can substitute the formal parameter with the true parameter

Analogy in C:

Abstraction: `int f (int x) {return x}`

Application: `f(y) evaluates to y`

λ-Calculus Evaluation (Informal)

*Observation: we can substitute the formal parameter with the true parameter (**with one exception!**)*

$$(\lambda x.\, (\lambda y.\, x))\ y = \lambda y.\, y\ ??$$

β-Reduction

Substitute x with t_2

$$(\lambda x. t_1)\ t_2 = t_1\{t_2/x\}$$

- When no free variable in t_2 is bound in t_1, we have $(\lambda x. t_1)\ t_2 = t_1\{t_2/x\}$
- Otherwise, apply α-reduction to t_1 until the condition above holds

$$(\lambda x. (x\ x))\ y \qquad = y\ y$$

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

β-Reduction Example

Example 1: $(\lambda x.\,(x\ x))\ y$

In this case, $(x\ x)$ corresponds to t_1,

y corresponds to t_2

FV(t_2) = y. y is not in t_1, hence, not bound

The first rule of substitution applies, which gives

$(\lambda x.\,(x\ x))\ y = y\ y$

β-Reduction Example

Example 2: $(\lambda x.(\lambda y.x))\ y$

In this case, $(\lambda y.x)$ corresponds to t_1,

y corresponds to t_2

FV(t_2) = $\{y\}$. y is used in t_1, we compute

FV(t_1) = $\{x\}$

So y is not free, hence, is bound in t_1. The first rule does not apply. We replace y with z in t_1 $(\alpha \text{ reduction})$

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

Then it's easy to check the first rule applies, so

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

β-Reduction

Substitute x with t_2

$$(\lambda x.\, t_1)\ t_2 = \ t_1\{t_2/x\}$$

- When no free variable in t_2 is bound in t_1, we have $(\lambda x.\, t_1)\ t_2 = \ t_1\{t_2/x\}$

=> Substitution is ***always correct*** when t_2 is a *closed term* (a term with no free variable)

β-Reduction Example

Ω Combinator: $\lambda x.\,(x\ x)$

Evaluate: $(\lambda x.(x\ x))\ (\lambda v.\,v)$ $= (\lambda v.\,v)\ (\lambda v.\,v) = (\lambda v.\,v)$

And $(\lambda x.(x\ x))\ (\lambda x.\,(x\ x))$

$= (\lambda x.(x\ x))\ (\lambda x.\,(x\ x))$ Infinite loop!

β-reduction rules for $(\lambda x.\,t_1)\ t_2$

Substitute x with t_2 in t_1 when t_2 is a *closed term* (a term with no free variable)

Another Useful Rule

$(\lambda x_1\ x_2\ \ldots xn. t)\ t_1\ t_2 \ldots\ t_n = t\{t_1/x_1\} \ldots \{t_n/x_n\}$
when $t_1\ t_2\ \ldots\ t_n$ are all closed terms

Evaluation

An ***evaluation*** of a lambda term is a sequence

$$e_1 = e_2 = e_3 = \ \dots$$

where each step is either an α-reduction or a β-reduction

Evaluation Order

No reduction order is specified in classical λ-Calculus

If evaluation terminates, any order gives same result

$$\begin{aligned} & (\lambda x.\, (\lambda y.\, x)\ z)\ u \\ = {} & (\lambda y.\ u)\ z \\ = {} & \ u \end{aligned}$$

$$\begin{aligned} & (\lambda x.\, (\lambda y.\, x)\ z)\ u \\ = {} & (\lambda x.\ x)\ u \\ = {} & \ u \end{aligned}$$

Is the λ-Calculus Turing complete?

Encoding: Boolean

Booleans

Shorthand for $\lambda x. (\lambda y. x)$

$$\mathrm{TRUE} \triangleq \lambda x\ y. x \qquad\qquad \mathrm{FALSE} \triangleq \lambda x\ y. y$$

Encoding of “if”?

Goal: $$\mathrm{IF}\ b\ t\ f = \begin{cases} t \text{ when } b \text{ is TRUE} \\ f \text{ when } b \text{ is FALSE} \end{cases}$$

Definition $\mathrm{IF} \triangleq \lambda b\ t\ f. \ (b\ t\ f)$

Encoding: Boolean

Booleans

$$\text{TRUE} \triangleq \lambda x\ y.\ x \qquad \text{FALSE} \triangleq \lambda x\ y.\ y$$

Encoding of "and"?

Goal: $$\text{AND}\ b_1\ b_2 = \begin{cases} \text{TRUE when } b_1, b_2 \text{ are both TRUE} \\ \text{FALSE otherwise} \end{cases}$$

Definition $\text{AND} \triangleq \lambda b_1\ b_2.\ (b_1\ (b_2\ \text{TRUE FALSE})\ \text{FALSE})$

Check that $\text{AND TRUE FALSE} = \text{FALSE (Note 2)}$