Winter 2020 Zi Hao He

CS442 Notes

1 Functional Programming

- to begin: Functional Programming in general, not a particular language
- need a model
    - simple
    - powerful enough to be useful

Church-Turing Thesis: Any algorithm can be simulated on a Turing machine.

Alan Turing

- Turing machine
- not an inspiring model for programming

Alonzo Church

- λ-calculus
- equivalent to Turing Machines
- the basis for all of functional programming

1.1 Untyped Lambda-Calculus

Calculus

- a system or method of calculation
- syntax + rules for manipulating syntax

Lambda-Calculus (1934) - "a calculus of functions"

Syntax: Abstract Syntax

$$
\begin{aligned}
& \quad variable, abstraction, application \\
<expr> &::= <var> \mid <abs> \mid <app> \\
<var> &::= a|b|c \\
& \quad \text{variable and body} \\
<abs> &::= \lambda <var> . <expr> \\
& \quad \text{rator and rand} \\
<app> &::= <expr><expr>
\end{aligned}
$$

Use parenthesis to disambiguate parses

Conventions:

1. Abstractions extend as far to the right as possible

   e.g. $\lambda x.y\ z = \lambda x.(y\ z) \neq (\lambda x.y)\ z$

2. Applications associate left-to-right

   e.g. $x\ y\ z = (x\ y)\ z \neq x\ (y\ z)$

Interpretation

- Var: Self explanatory
- Abs: $\lambda x.E =$ "function" taking argument x and returning expr E
- App: $M\ N$ - result of applying "function" M to argument N

Consider:

$\lambda x.x$ - Here the first x is a binding occurrence and second x is a bound occurrence.

$\lambda x.x\ y\ x$ - The y is a free occurrence (fully parenthesized function is $\lambda x.((x\ y)\ x))$

$\lambda x.x\ (\lambda x.x)\ x$ - Same variable name might be bound in difference places as long as the scope is not clashing

$x\ \lambda x.x$ - The x in the front is free while the other one is bound

Informally:

A variable is bound if it has a bound occurrence.

A variable is free if it has a free occurrence

The same variable can be used in both bound and free occurrences

Formally:

Definition: Let E be an expression. The bound variables of E, $B \cup [E]$ are given by

$B \cup [x] = \emptyset$

$B \cup [\lambda x.E] = B \cup [E] \cup \{x\}$

$B \cup [M\ N] = B \cup [M] \cup B \cup [N]$

x is bound in E if $x \in B \cup [E]$

The free variables of E, $F \cup [E]$ are given by

$F \cup [x] = \{x\}$

$F\cup[\lambda x.E] = F\cup[E] \setminus \{x\}$

$F\cup[M\ N] = F\cup[M] \cup F\cup[N]$

x is free in E if $x \in F\cup[E]$

A variable can be both bound and free: $F\cup[x\ \lambda x.x] = B\cup[x\ \lambda x.x] = \{x\}$

But each occurrence is either bound or free, not both

Two occurrences of a variable x "mean the same thing" if

1. they are both free occurrences
2. OR they have the same binding occurrence

$\lambda x.\lambda y.y\ x$ and $\lambda a.\lambda b.b\ a$ these should mean the same thing

$\lambda x.y\ x$ and $\lambda x.w\ x$ should not mean the same thing because y and w do not have to be the same free variable

You can change the names of bound variables, but not free ones.

Formally:

Definition (α-conversion): For all x, y, M, $\lambda x.M =_\alpha \lambda y.M[y/x]$ if $y \notin F\cup[M]$

$M[N/x]$ = "substitute N for x in M"

More generally, if $C[\lambda x.M]$ denotes an expr in which $\lambda x.M$ occurs as a subexpression, then $C[\lambda x.M] =_\alpha C[\lambda y.M[y/x]]$ if $y \notin F\cup[M]$.

e.g.

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

$x\ \lambda x.x =_\alpha x\ \lambda a.a$

$\lambda a.b\ a =_\alpha \lambda c.b\ c \neq_\alpha \lambda b.b\ b \neq_\alpha \lambda a.d\ a$

Computation:

Expr. $(\lambda x.M)\ N$ is called a (β-)redex (reductible expression)

We expect: $(\lambda x.M)\ N \Rightarrow$ evaluate M, with N substituted for x i.e. $M[N/x]$

Definition (β-reduction): For all M, N, x, $(\lambda x.M)\ N \rightarrow_\beta M[N/x]$

More generally - for all contexts C, $C[(\lambda x.M)\ N] \to_\beta C[M[N/x]]$

$\to_\beta$ is a binary relation on terms

$A \to_\beta B$: A beta reduces to B in one step

$A \to_\beta^n B$: A beta reduces to B in n steps

$A \to_\beta^* B$: A beta reduces to B in 0 or more steps

$A \to_\beta^+ B$: A beta reduces to B in 1 or more steps

Evaluating $M[N/x]$

Want: Substitute N for all free occurrences of x in M

Definition (substitution, naive(wrong)):

$x[E/x] = E$

$y[E/x] = y$, $y \neq x$

$(M\ N)[E/x] = M[E/x]\ N[E/x]$

$(\lambda x.P)[E/x] = \lambda x.P$

$(\lambda y.P)[E/x] = \lambda y.P[E/x]$, $y \neq x$

E.g.

$$\begin{aligned}(\lambda x.x\ y)\ z &\to_\beta (x\ y)[z/x] \\ &= x[z/x]\ y[z/x] \\ &= z\ y\end{aligned}$$

$$(\lambda x.x)\ a \to_\beta x[a/x] = a$$

$$\begin{aligned}(\lambda x.\lambda y.x)\ a\ b &\to_\beta (\lambda y.x)[a/x]\ b \\ &= (\lambda y.x[a/x])\ b \\ &= (\lambda y.a)\ b \\ &\to_\beta a[b/y] \\ &= a\end{aligned}$$

$$\begin{aligned}(\lambda x.\lambda y.y)\ a\ b &\to_\beta (\lambda y.y)[a/x]\ b \\ &= (\lambda y.y[a/x])\ b \\ &= (\lambda y.y)\ b \\ &\to_\beta y[b/y] \\ &= b\end{aligned}$$

Consider:

$$\begin{aligned}(\lambda x.\lambda y.x)\ y\ w &\rightarrow_\beta (\lambda y.x)[y/x]\ w \\ &= (\lambda y.x[y/x])\ w \\ &= (\lambda y.y)\ w \\ &\rightarrow_\beta y[w/y] \\ &= w\end{aligned}$$

We can see from this that the last rule of $(\lambda y.P)[E/x] = \lambda y.P[E/x]$, $y \neq x$ must be wrong.

What happened? Free variable y became bound after substitution

As a result, the binding occurrence of x changed

$\lambda x.\lambda y.x \leftarrow$ the inner x is bound to the λx on the outside.

This is called **Dynamic binding** - meaning of variables uncertain until runtime

We want **static binding** - meanings of variables fixed before runtime

Definition (substitution, fixed):

$x[E/x] = E$

$y[E/x] = y$, $y \neq x$

$(M\ N)[E/x] = M[E/x]\ N[E/x]$

$(\lambda x.P)[E/x] = \lambda x.P$

$(\lambda y.P)[E/x] = \lambda y.P[E/x]$, $y \notin F \cup [E]$

What if y is free???? (Thursday class)