CSE114A lecture 4

agenda:

- ✓ – Variable scope
- ✓ – Free and bound variables
- ✓ – Computing the free variables of an expression
- ✓ – Revisiting the beta rule
- ✓ – Capture-avoiding substitution
- ✓ – Break / quiz
- ✓ – More λ-calc encoding: Pairs!

Variable scope

$\backslash x \to e$

↑ binder (points to $\backslash x$)

binding: association of a name to... something!

Scope of a binding (↙ name-entity association):
the part of a program where the name can be used to refer to the entity.

$(\backslash x \to \boxed{e})\ y\ \underline{x}$

- body (the boxed e)
- Occurrences of x in here are bound by this binder
- not bound by any binder (the last x)

in a function

$\backslash x \to e$

any occurrence of x in e is bound by the binder $\backslash x$.

$\backslash x \to x$

$\backslash x \to (\backslash y \to x)$

An occurrence of a variable is free if it's not bound.

$x\ y$ ← x and y occur free

$\backslash y \to x\ y$ ← x occurs free, y only occurs bound

$(\backslash y \to x\ y)\ y$ ← x occurs free, and we have free and bound occurrences of y.

Computing the free variables of an expression

$e ::= x \mid \lambda x.\ e \mid e_1\ e_2$

let $FV(e)$ for the set of variables that occur free in e.

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

$FV(\backslash x \to e) = FV(e) - \{x\}$ ← set difference

$FV(e_1\ e_2) = FV(e_1) \cup FV(e_2)$

That's it!

An expression with no free variables is closed.

A closed expression is also known as a combinator.

TRUE, FALSE, ITE, ZERO, ONE, etc. are all combinators.

$\backslash x \to x$ is the simplest combinator.

Revisiting the β rule

$(\backslash x \to e_1)\ e_2 \to_\beta e_1[x := e_2]$

This means "e_1, but with free occurrences of x replaced with e_2".

But...

$(\backslash x \to (\backslash x \to x))\ y$

$=b> \backslash x \to x$

But if we just replaced occurrences of x with y, we'd get $\backslash x \to y$!

So really, $e_1[x := e_2]$ should mean "e_1, but with all free occurrences of x in it replaced with e_2."

But... is that all?

$(\backslash x \to (\backslash y \to x))\ y$

$=b> \backslash y \to y$

is that right?
Why did this happen?

$(\backslash x \to (\backslash y \to x))$ is equivalent to $(\backslash x \to (\backslash z \to x))$

What went wrong here is that y occurred free in the argument, and then it got captured by the $\backslash y$ binder in the body of the function!

So, let's revise what $e_1[x := e_2]$ means once again.
it should mean:
"e_1, but with all free occurrences of x replaced with e_2, as long as no variables that occur free in e_2 get captured by binders in e_1."

So, sometimes we'll need to rename formal parameters to be able to take a β step!

The quiz question:

$(\backslash x \to x\ (\backslash y \to z))\ (x\ y)$

(x: bound; z: free; x, y: free free)

$\{x, y, z\}$

$(\backslash x \to e_1)\ e_2 \to_\beta e_1[x := e_2]$

$(\backslash x \to x\ (\backslash y \to z))\quad (x\ y)$

($\backslash x$; $e_1 = x\ (\backslash y \to z)$; $e_2 = (x\ y)$)

$=b> (x\ y)\ (\backslash y \to z)$

Capture-avoiding substitution

$(\backslash x \to e_1)\ e_2 \to_\beta e_1[x := e_2]$

How to define $e_1[x := e_2]$?

$e ::= x \mid \backslash x \to e \mid e'\ e''$

We need to deal with the cases where e_1 is:

- a variable ✓
- a lambda abstraction
- an application ✓

e_1 might be a variable, and it might be the same as the binder, or not!

$\boxed{e_1}[x := e_2]$

$x[x := e_2] = e_2$!

$y[x := e_2] = y$

$(e'\ e'')[x := e_2] = (e'[x := e_2]\ e''[x := e_2])$

↑ Just do the substitution on both subexpressions.

$(\backslash x \to e')[x := e_2] = \backslash x \to e'$

There are no free occurrences of x, so this just stays the same! yay.

$(\backslash y \to e')[x := e_2]$ = if y is not in $FV(e_2)$: $\backslash y \to (e'[x := e_2])$

(a function with anything other than x as its formal parameter)

but if y is in $FV(e_2)$, substitution is undefined! and we have to do a renaming step.

$(\backslash x \to (\backslash y \to x))\ y$

Before taking a beta step, we have to rename the formal parameter y and replace all occurrences of y that were bound by it with the new name.

e.g.:

$(\backslash x \to (\backslash z \to x))\ y$

This is called an α-step, or α-renaming, (← greek letter alpha) written =a> in Elsa.

$\backslash x \to e$ can be renamed, e.g., to $\backslash z \to e$ as long as the new name you pick (here, z) doesn't occur free in e.

More λ-calc encoding: Pairs

A pair is a two-element tuple.
What can you do with a pair?

- take out first element
- take out second element
- construct pair out of two elements

FST = $\backslash p \to$?

SND = $\backslash p \to$?

PAIR = $\backslash x\ y \to$?