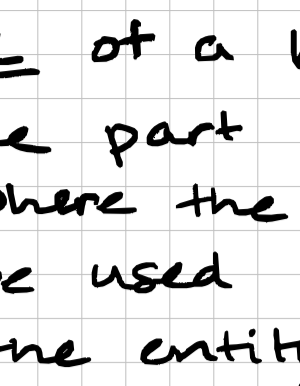


CSE1114A lecture 4

agenda:

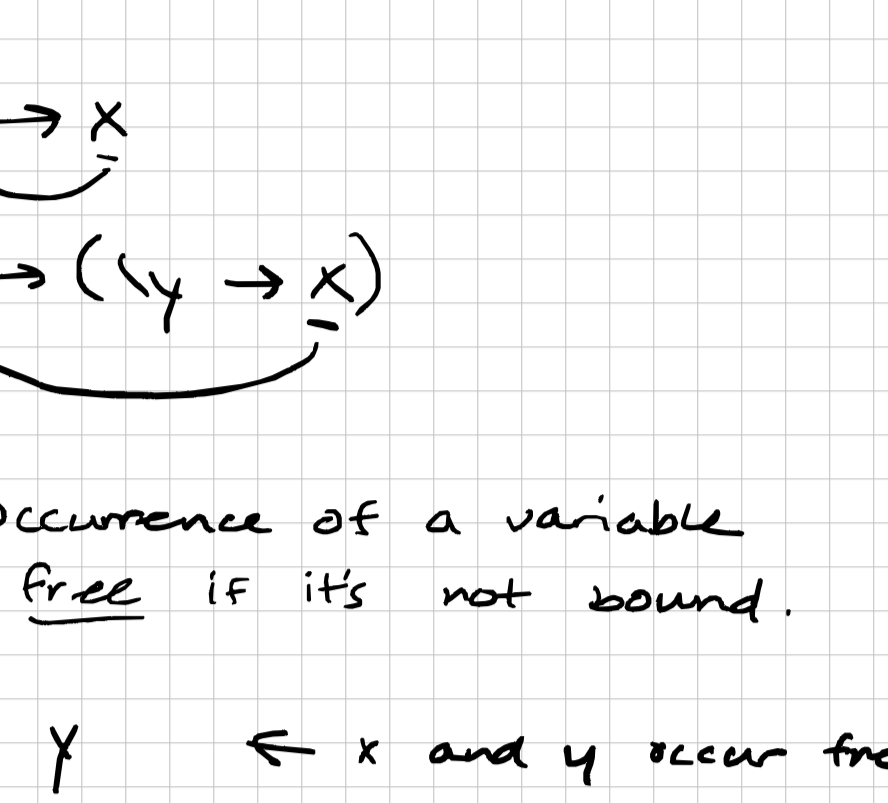
- ✓ Variable scope
- ✓ Free and bound variables
- ✓ Computing the free variables of an expression
- ✓ Revisiting the beta rule
- ✓ Capture-avoiding substitution
- ✓ Break/quiz
- ✓ more λ -calc encoding: Pairs!

Variable scope



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

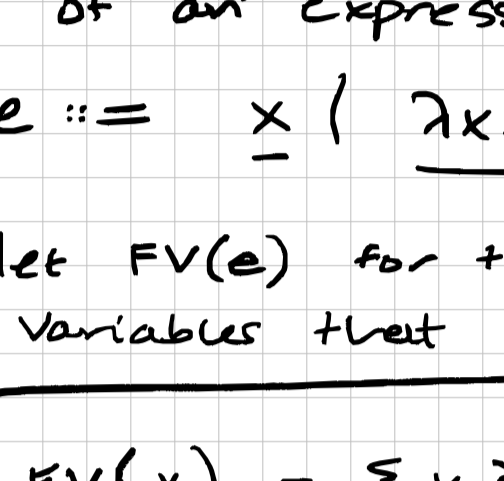
Scope of a binding: ^{name-entity association} the part of a program where the name can be used to refer to the entity.



in a function

$$\lambda x \rightarrow e$$

any occurrence of x in e is bound by the binder λx .



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

x y ← x and y occur free

$\lambda y \rightarrow x y$ ← x occurs free, y only occurs bound

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

Computing the free variables of an expression

$$e ::= \underline{x} \mid \underline{\lambda x. e} \mid \underline{e_1 e_2}$$

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

$FV(x) = \{x\}$ ↙ set difference
 $FV(\lambda x \rightarrow e) = FV(e) - \{x\}$
 $FV(e_1 e_2) = FV(e_1) \cup FV(e_2)$
 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.

$\lambda x \rightarrow x$ is the simplest combinator.

Revisiting the β rule

$$(\lambda x \rightarrow e_1) e_2 \rightarrow_{\beta} e_1[x := e_2]$$

This means "e₁, but with ^{free} occurrences of x replaced with e₂".

But...

$$(\lambda x \rightarrow (\lambda x \rightarrow x)) y$$

=> $\lambda x \rightarrow x$

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

So really, $e_1[x := e_2]$ should mean

"e₁, but with all free occurrences of x in it replaced with e₂."

But... is that all?

$$(\lambda x \rightarrow (\lambda y \rightarrow x)) y$$

=> $\lambda y \rightarrow y$

is that right?

Why did this happen?

$(\lambda x \rightarrow (\lambda y \rightarrow x))$ is equivalent to $(\lambda x \rightarrow (\lambda z \rightarrow x))$

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

So, let's revise what $e_1[x := e_2]$ means once again.

it should mean:

"e₁, but with all free occurrences of x replaced with e₂, as long as no variables that occur free in e₂ get captured by binders in e₁."

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

The quiz question:

$$(\lambda x \rightarrow e_1) e_2 \rightarrow_{\beta} e_1[x := e_2]$$

$$\frac{(\lambda x \rightarrow (x (\lambda y \rightarrow z))) (x y)}{\lambda x \quad e_1 \quad e_2}$$

=> $(xy) (\lambda y \rightarrow z)$

Capture-avoiding substitution

$$(\lambda x \rightarrow e_1) e_2 \rightarrow_{\beta} e_1[x := e_2]$$

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

$$e ::= x \mid \lambda x \rightarrow e \mid e' e''$$

We need to deal with the cases where e₁ is:

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

e₁ might be a variable, and it might be the same as the binder, or not!

$$[e_1][x := e_2]$$

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

$$y[x := e_2] = y$$

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

↑ ↑
Just do the substitution on both subexpressions.

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

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

$$(\lambda y \rightarrow e')[x := e_2] = \text{if } y \text{ is not in } FV(e_2):$$

a function with anything other than x as its formal parameter $\lambda y \rightarrow (e'[x := e_2])$

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

$$(\lambda x \rightarrow (\lambda y \rightarrow x)) y$$

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

This is called an α -step, or α -renaming, ^{↑ greek letter alpha} written => in Elsa.

$\lambda x \rightarrow e$ can be renamed, e.g., to $\lambda z \rightarrow e$ 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

$$FST = \lambda p \rightarrow ?$$

$$SND = \lambda p \rightarrow ?$$

$$PAIR = \lambda x y \rightarrow ?$$