

CSE114A lecture 6

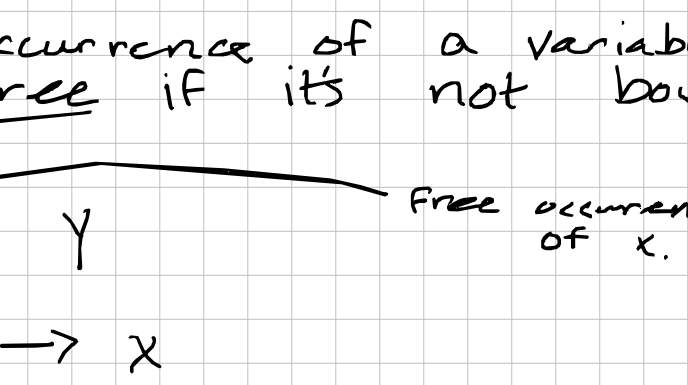
Agenda:

- Recap: Variable scope, Free and bound variables ✓
- Computing the set of Free variables of an expression ✓
- Revisiting the β -rule ✓
- If time: capture-avoiding substitution
- If time: Decrementing numbers!

Variable scope

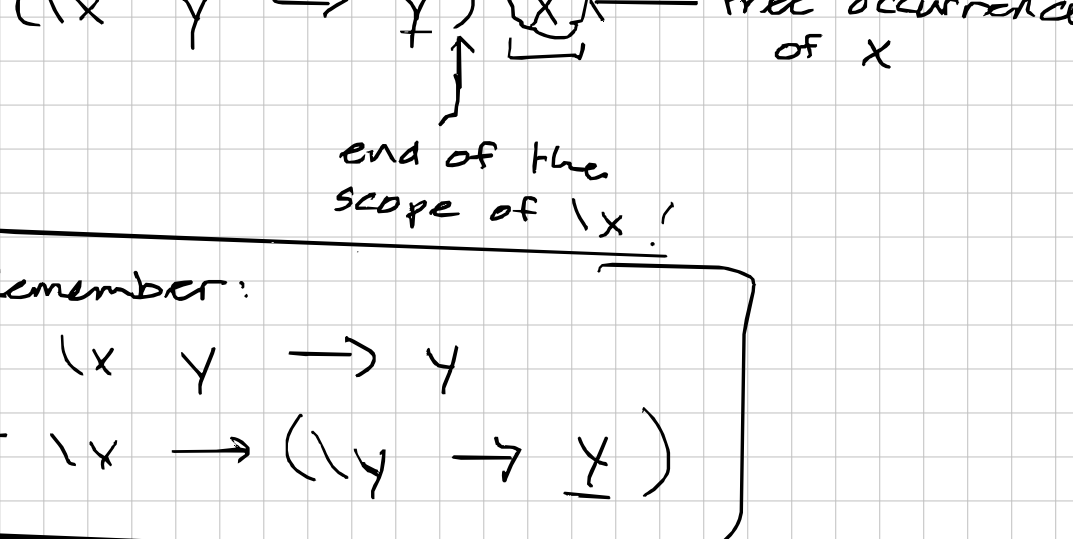
A binding is an association of a name to some entity.

The scope of a binding in a program is the part of the program where the name can be used to refer to the entity.



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

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

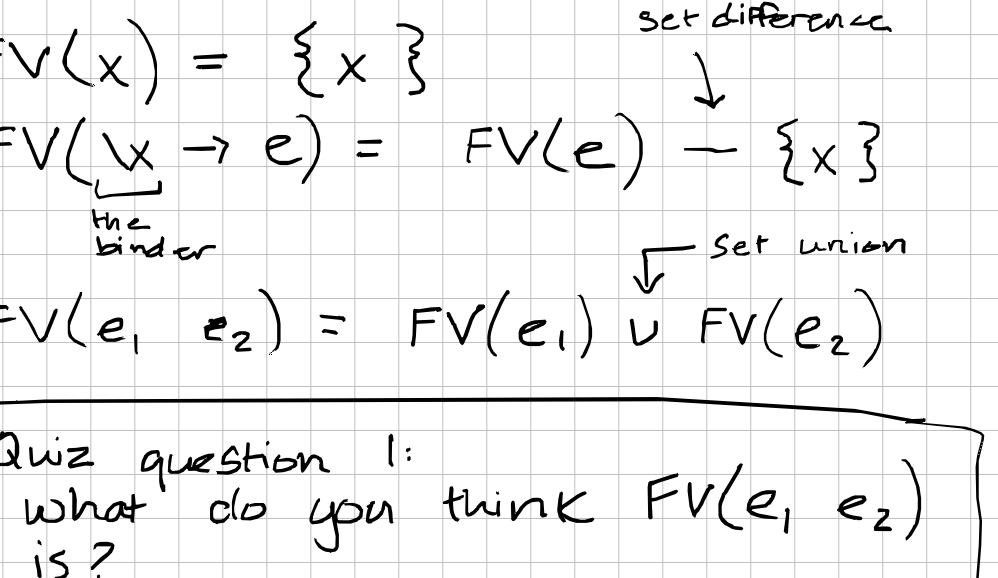


Remember:

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

Computing the set of Free variables of an expression

(without evaluating the expression!)



Let's define $FV(e)$ to be the set of variables that occur free in e .

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

$$FV(\underbrace{\lambda x}_{\text{the binder}} \rightarrow e) = FV(e) - \{x\}$$

set difference

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

set union

Quiz question 1: what do you think $FV(e_1 \ e_2)$ is? Hint: you can use set operations

What we just did was a static analysis of e !
 (without running the program)

An expression with no free variables is said to be closed.

A closed expression is also known as a combinator.

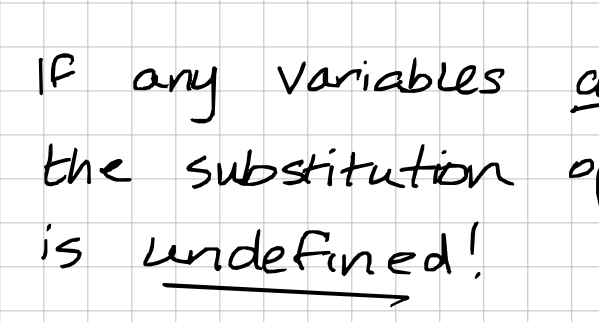
Quiz question 2: write the simplest combinator you can think of!

$$\lambda x \rightarrow x$$

Revisiting the beta rule

$$(\lambda x \rightarrow e_1) \ e_2 \Rightarrow e_1[x := e_2]$$

What if you had...

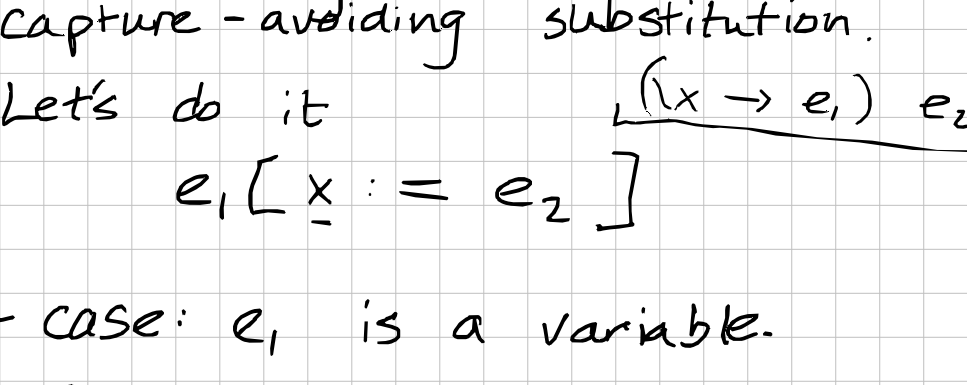


What should we get?

If we replaced all occurrences of x in the function's body with y , we'd get $\Rightarrow \lambda x \rightarrow y$ which isn't right!

What we really want is for $e_1[x := e_2]$ to mean "replace all free occurrences of x with e_2 in e_1 ".

wait... is that it? what about this?



What went wrong? y occurred free in the argument, but it got captured by the λy binder in the body of the function!

So we need to be even more subtle about what $e_1[x := e_2]$ means:

It means: substitution e_1 , but with all free occurrences of x in e_1 replaced with e_2 , as long as no variables that occur free in e_2 get captured by binders in e_1 .

If any variables do get captured, the substitution operation is undefined!

so if we have an expression like $\lambda x \rightarrow (\lambda y \rightarrow x)$ and we want to apply it to y , we have to rename the formal parameter y in the binder λy so that we can take a \Rightarrow step.

E.g. we could rename y to z .

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

rename:

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

now step:

$$\lambda z \rightarrow y$$

we can, if we want, now formally define capture-avoiding substitution.

Let's do it $(\lambda x \rightarrow e_1) \ e_2 \Rightarrow e_1[x := e_2]$

- case: e_1 is a variable.

if e_1 is x : $(\lambda x \rightarrow x) \ e_2 \Rightarrow e_2$

otherwise (e_1 isn't x): $(\lambda x \rightarrow y) \ e_2 \Rightarrow y$

- case: e_1 is an application $e_3 \ e_4$:

$$(e_3 \ e_4)[x := e_2] = (e_3[x := e_2]) \ (e_4[x := e_2])$$

- case: e_1 is a function $\lambda x \rightarrow e_3$ with x as its formal parameter:

$$(\lambda x \rightarrow e_3)[x := e_2] = \lambda x \rightarrow e_3$$

no free occurrences of x in here!

Quiz question 3: what's $(\lambda x \rightarrow e_1) \ e_2 \Rightarrow ?$
 $(\lambda x \rightarrow e_3)[x := e_2] \ ?$

Finally, if e_1 is a function with something other than x as its formal parameter:

$$(e_1 = \lambda y \rightarrow e_3)$$

we have to check if the formal parameter is actually in the free variables of e_2 .