CSCI 334:
Principles of Programming Languages

Lecture 8: Lambda, lambda, lambda!

Instructor: Dan Barowy

Williams

Topics

Lambda calculus—how to survive it

Your to-dos

1. Lab 4, **due Wednesday 3/5** (solo lab)
2. Read *Syntax* and *Introduction to the Lambda Calculus, Part 1*.

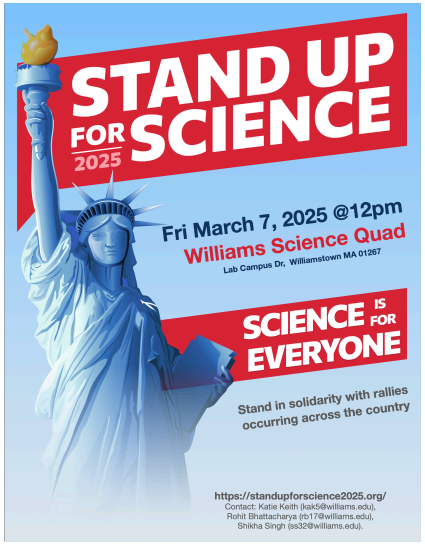
Announcements

- CS Colloquium this **Friday, Mar 5 @ 2:35pm in Wege Auditorium (TCL 123)**

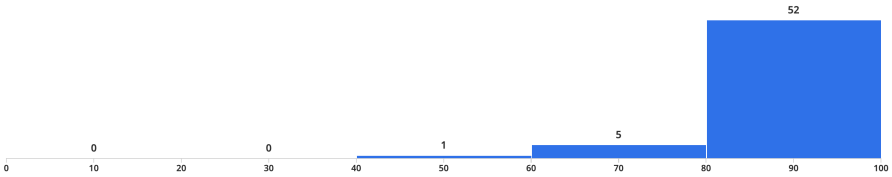
Scalable Data Systems
Prof. Prashant Pandey (Northeastern)

Prashant builds scalable data systems with strong theoretical guarantees and employs them to democratize next-generation data analyses. His work applies to high-performance computing, computational biology, stream processing, and storage.

Announcements



Quiz 3



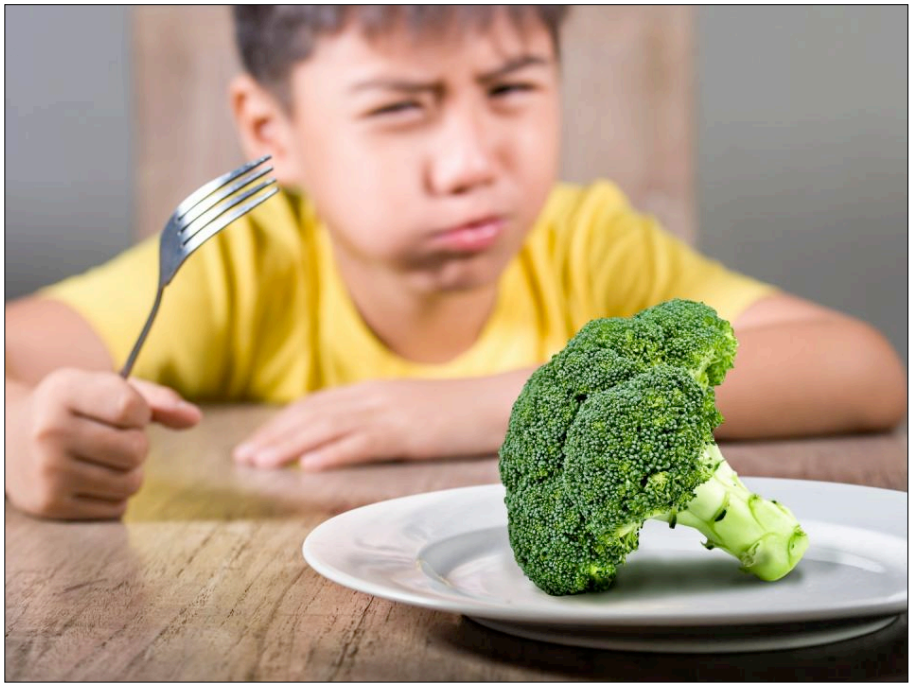
Median	Maximum	Mean	Std Dev
95.0	100.0	91.79	12.26

Wow, great work!

If you got <80%, let's find a time to meet.

Announcements

- **Midterm exam**, **Thursday, March 20**, on paper, in class.
- Resubmissions are due by the **last day of the final exam reading period**.



β-Reduction

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

How we “call” or **apply** a function to an argument

Rule:

$[\![(\lambda x.\langle expr\rangle)\,y]\!] =_\beta [\![[y/x]\langle expr\rangle]\!]$

Let’s reduce this

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

β-Reduction

$(\lambda x.x)\,x$	given
$(\lambda y.[y/x]x)\,x$	α-reduce y for x
$(\lambda y.y)\,x$	inner α-reduction
$[x/y]y$	β-reduce x for y
x	done

Watch out!

$(\lambda x.\lambda x.x)\,x$	given
$([x/x]\lambda x.x)$	β-reduce x for x
$(\lambda x.x)$	β-reduce inner expr
	done

The inner lambda term **redefines** x and therefore “blocks” substitution of x.

How far do we go?

We keep going until there is **nothing left to simplify**.

`x`	← done
`xx`	← done
`λx.y`	← done
`(λx.xy)z`	← not done

That "most simplified" expression is called a **normal form**.

An expression that can be simplified is a called a **redex**.

Try this one with a partner

`(λx.λy.yx)xy`

(don't forget precedence/associativity rules)

Example

`(λa.λb.(- a b)) 2 1`

Abstract Syntax Trees

An **abstract syntax tree** (**AST**) is a tree representation of a language such that **all operations are interior nodes** and **all data are leaf nodes**. As such, ASTs are frequently used to represent programs.

An AST can be obtained from a derivation by a set of **tree-transformation rules**. These rules are language-specific. **See handout**.

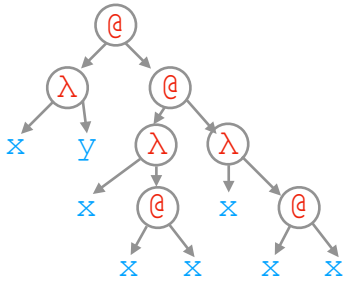
Activity: Abstract Syntax Trees

Derive this expression, and then **convert** it to an AST.

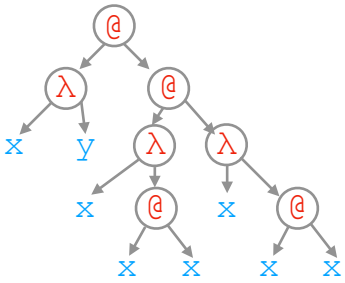
`(λx.y)((λx.xx)(λx.xx))`

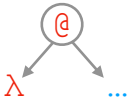
Activity: Abstract Syntax Trees

Did you get this AST?



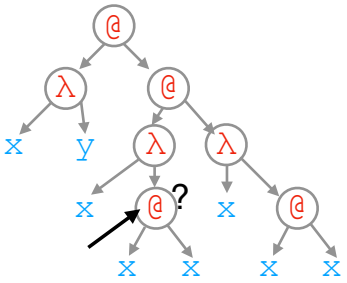
Which reduction do I perform?

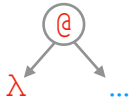


Redex: 

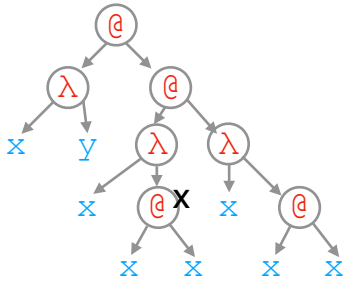
redex = application with abstraction as left child.

Which reduction do I perform?

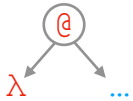


Redex: 

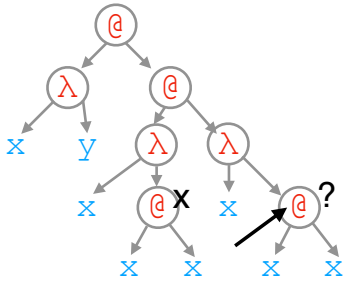
Which reduction do I perform?



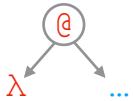
Redex:



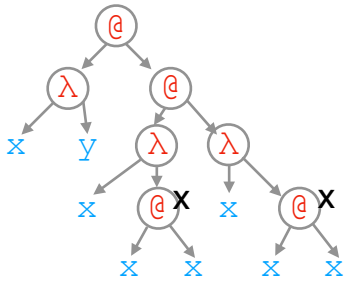
Which reduction do I perform?



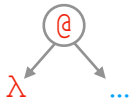
Redex:



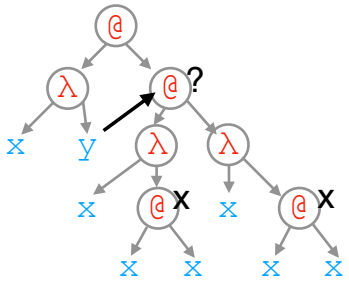
Which reduction do I perform?



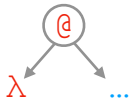
Redex:



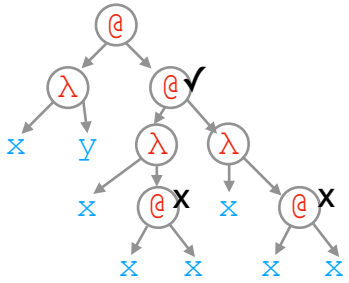
Which reduction do I perform?



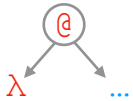
Redex:



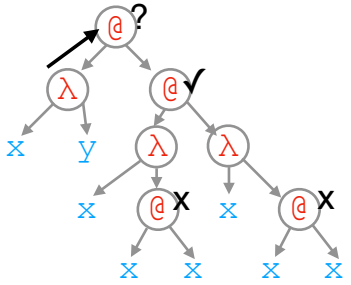
Which reduction do I perform?



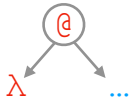
Redex:



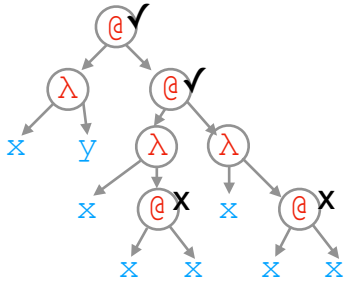
Which reduction do I perform?



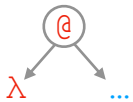
Redex:



Which reduction do I perform?



Redex:



Reduction strategies

`(λx.y) ((λx.xx) (λx.xx))`

function argument

Reduction strategies

(λx.y) ((λx.xx) (λx.xx))

function argument

Which reduction do I perform?

(λx.y) ((λx.xx) (λx.xx))

function argument

(λx.y) ((λx.xx) (λx.xx))

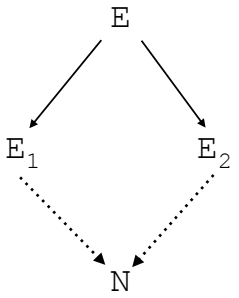
function argument

Two well-known reduction orders

- Normal order
- Applicative order

Sometimes multiple reductions available

Order (mostly) does not matter



If $E \rightarrow E_1$ and $E \rightarrow E_2$
then $E_1 \rightarrow^* N$ and $E_2 \rightarrow^* N$
for some N

confluence

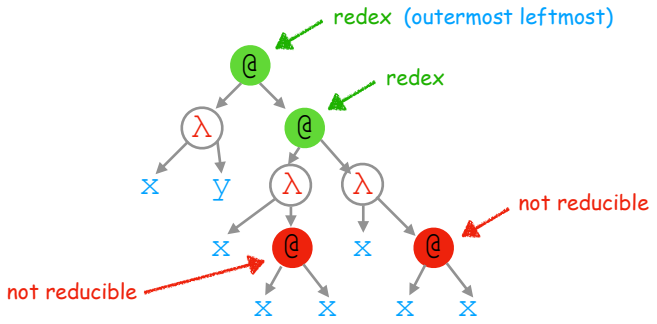
Demonstration

Normal order ("outermost leftmost") reduction

`(λx.y) ((λx.xx) (λx.xx))`

What does "outermost leftmost" mean?

`(λx.y) ((λx.xx) (λx.xx))`



What does "outermost leftmost" mean?

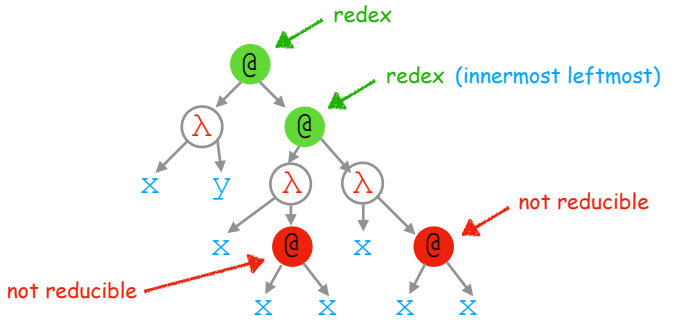
Demonstration

Applicative order ("innermost leftmost") reduction

`(λx.y) ((λx.xx) (λx.xx))`

What does "innermost leftmost" mean?

`(λx.y) ((λx.xx) (λx.xx))`



What does "innermost leftmost" mean?

Meaning of "equivalence"

The only **equivalent** expressions in the lambda calculus are those that are **textually identical**.

$\lambda a.aa \neq \lambda b.bb$

after alpha reducing a for b:

$\lambda a.aa = \lambda a.aa$

Recap & Next Class

Today:

More lambda reductions

Next class:

Computability