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

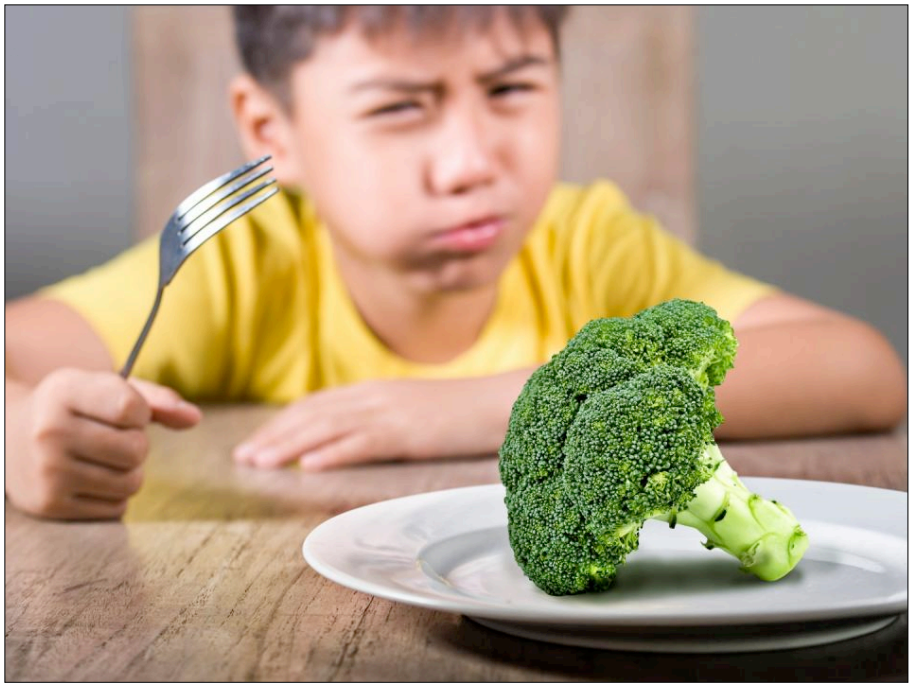
1. Lab 4, **due Sunday 10/8** (partner lab)
2. Review handouts/feedback if you haven't already…

Announcements

- **Midterm exam**, **Thursday, October 19**, on paper, in class.
- Resubmissions are due by the **last day of the final exam reading period**.

Mountain Day, whenever that is…

- No office hours (faculty “retreat”)



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))

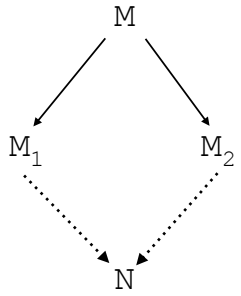
function argument

(λx.y) ((λx.xx) (λx.xx))

function argument

Sometimes multiple reductions available

Order (mostly) does not matter



If $M \rightarrow M_1$ and $M \rightarrow M_2$
then $M_1 \rightarrow^* N$ and $M_2 \rightarrow^* N$
for some N

"confluence"

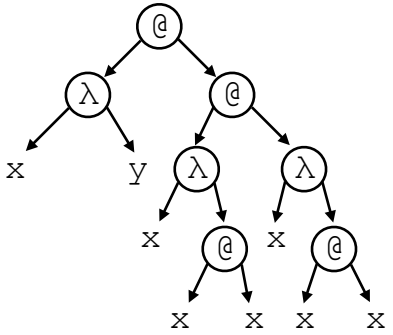
Demonstration

Normal order ("outermost leftmost") reduction

(λx.y) ((λx.xx) (λx.xx))

What does "outermost leftmost" mean?

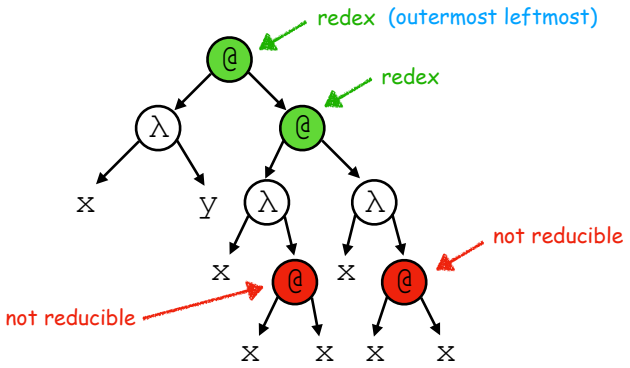
(λx.y) ((λx.xx) (λx.xx))



Abstract syntax tree.

(contains only operators and values)

`(λx.y) ((λx.xx) (λx.xx))`



Redex: application with abstraction as left child.

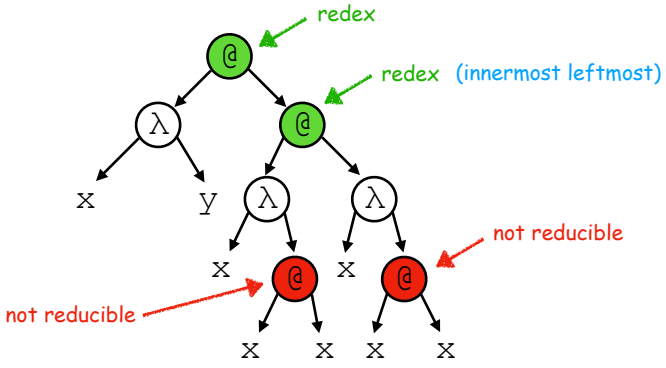
Demonstration

Applicative order ("innermost leftmost") reduction

`(λx.y) ((λx.xx) (λx.xx))`

What does "innermost leftmost" mean?

`(λx.y) ((λx.xx) (λx.xx))`



Redex: application with abstraction as left child.

Meaning of "equivalence"

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

`λa.aa ≠ λb.bb`

after alpha reducing a for b:

`λa.aa = λa.aa`

One caveat about reduction orders

Although reduction order "does not matter" (because the LC is confluent), only the **normal order** reduction is **guaranteed to terminate** for expressions that **have a normal form**.

(see LC, part 2 from packet for more detail)

More practice finding redexes

```
(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2
```

Normal order: `(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`

Applicative order: `(λa.(λz.(+ x z))(``(λz.(+ x z))`` `a` `)) 2`

Neither: `(λa.``(λz.(+ x z))`` `(```(λz.(+ x z))`` `a` `)) 2`

Trouble matching parens? Try this.

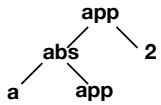
```
(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2
1   2    3     3 2 2 3   4     4 3   2 1
```

More practice finding redexes

```
(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2
```

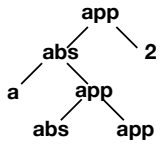
More practice finding redexes

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



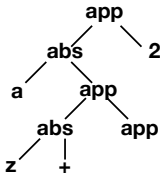
More practice finding redexes

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



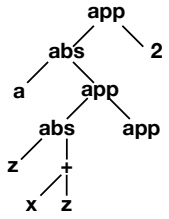
More practice finding redexes

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



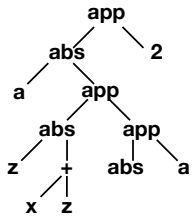
More practice finding redexes

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



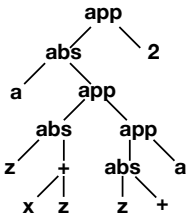
More practice finding redexes

```
(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2
```



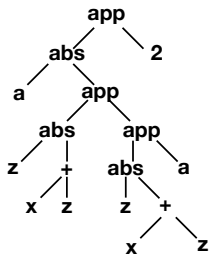
More practice finding redexes

```
(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2
```



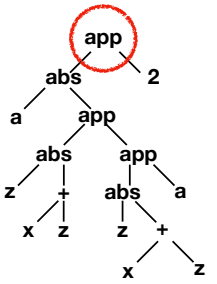
More practice finding redexes

```
(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2
```



More practice finding redexes

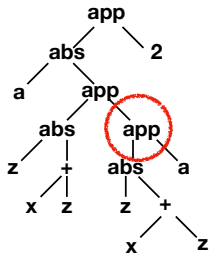
```
(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2
```



When I say *normal order*, I mean: “leftmost outermost” application

More practice finding redexes

```
(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2
```



When I say *applicative order* I mean: “leftmost innermost” application

Activity

Applicative order reduction:

```
(λf.λx.f(f x))(λz.(+ x z))2
```

Applicative order is “innermost leftmost” first.

Activity

Normal order reduction:

```
(λf.λx.f(f x))(λz.(+ x z))2
```

Normal order is “outermost leftmost” first.

Recap & Next Class

Today:

More lambda reductions

Next class:

Computability