CSCI 334:
Principles of Programming Languages

Lecture 9: Computability

Instructor: Dan Barowy

Williams

Topics

Last words about the lambda calculus

Function graphs

Decidability (maybe)

Your to-dos

1. Lab 5, **due Wednesday 3/12** (partner lab)
2. Start studying for the midterm

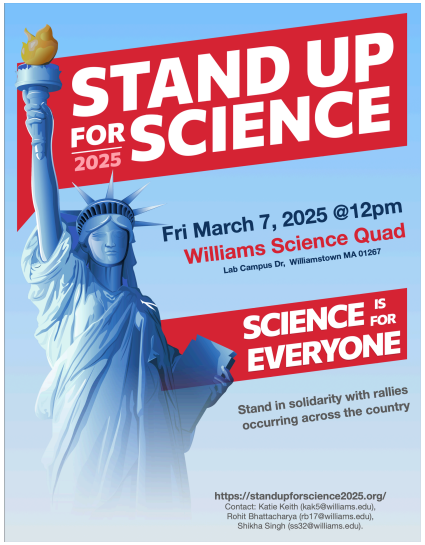
Announcements

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

Scalable Data Systems
Prof. Prashant Pandey (Northeastern)

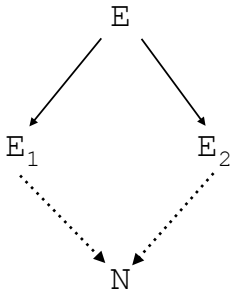
Prashant builds scalable data systems with strong theoretical guarantees and employs them to democratize next-generation data analyses. His work applies to high-performance computing, computational biology, stream processing, and storage.

Announcements



Recall: confluence

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

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)

Lambda Calculus with Addition Op

```
<expr>  ::= <value>
          |  <app>
          |  <abs>
          |  <par>
          |  <add>
<value> ::= v ∈ ℕ
          |  <var>
<var>   ::= α ∈ { a .. z }
<app>   ::= <expr><expr>
<abs>   ::= λ<var>.<expr>
<par>   ::= (<expr>)
<add>   ::= (+ <expr> <expr>)
```

Operations: `<app>`, `<abs>`, and `<add>`.

Data: `<value>`.

Trouble finding redexes? Try this.

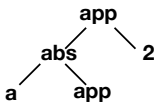
```
(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2
1   2   3     3 2 2 3   4     4 3   2 1
```

Or use the AST

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

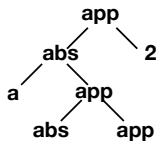
Or use the AST

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



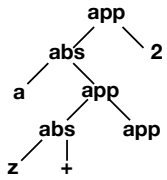
Or use the AST

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



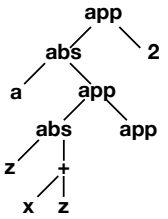
Or use the AST

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



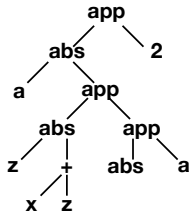
Or use the AST

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



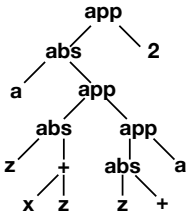
Or use the AST

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



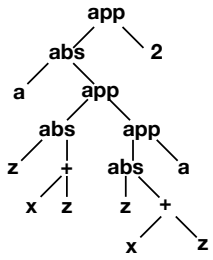
Or use the AST

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



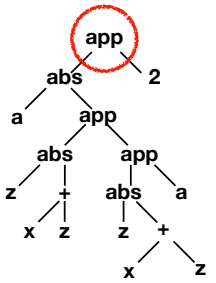
Or use the AST

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



Or use the AST

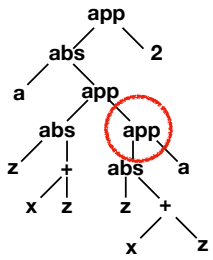
(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



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

Or use the AST

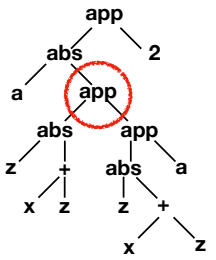
(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



When I say *applicative order* I mean: "**innermost leftmost**" application

Or use the AST

(λa.(λz.(+ x z))((λz.(+ x z)) a)) 2



Here's a **third** one! Neither normal nor applicative.

Which can I do? Any of them.

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

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

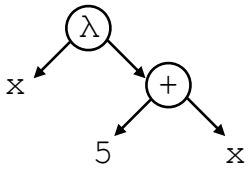
Applicative order: `(λa.(λz.(+ x z))((λz.(+ x z)) a )) 2`

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

Translating real code to lambda expressions

```
def f(x):
    return 5 + x
```

```
λx.(+ 5 x)
```

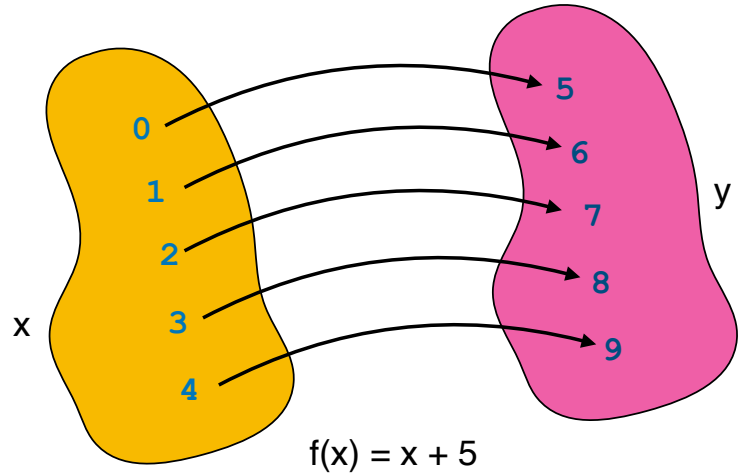


Practice this one on your own time

```
(λx.xx)(λy.yx)z
```

Computability

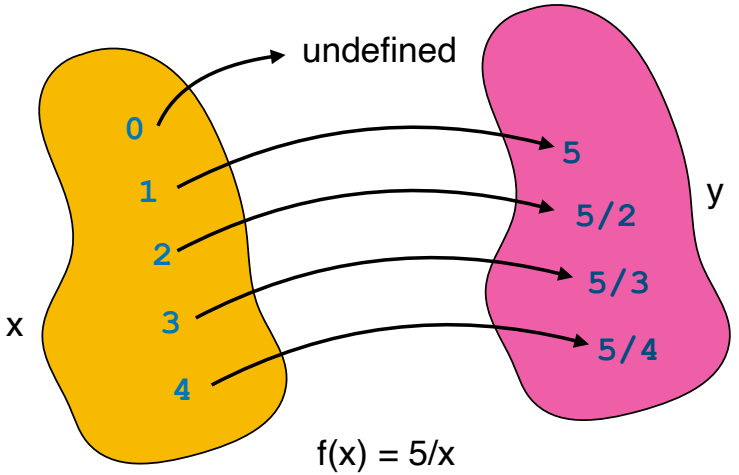
Intuition: total function



f(x) = x + 5

For every element in x, there is a corresponding element in y. x maps to at most one element in y.

Intuition: partial function



f(x) = 5/x

x still maps to at most one element in y, however, there is not a y for every x.

The **graph** of a function

f(x) = x + 5

{<x, x+5> | x ∈ ℤ}

{<x, x+5> | x is an integer}

The graph is **not a chart**!

It is a **set**, in set-builder notation.

The **graph** of a function

f(x) = 5/x

{<x, 5/x> | x ∈ ℤ ∧ x ≠ 0}

The graph is **not a chart**!

It is a **set**, in set-builder notation.

Undefinedness

x/0

Activity

Decidability Problems

A **decidability problem** is a question with a **yes** or **no** answer about an **input**.

"Is x prime?"

Restated: is there an **algorithm**?

Importantly, we want an algorithm that works for **all inputs in a domain**. We want a **total function**.

If there is **no total function**, then the problem is referred to as **undecidable**.

Decidability Problems

"Is x prime?"

What do you think? **Decidable** or **undecidable**?

The algorithm **does not** need to be efficient.

Computable Problems

An algorithm should return an answer in a **finite** amount of time.

Computability and **totality** are different things.

A problem is **decidable** if it is both computable and total.

The Halting Problem

Decide whether program **P** halts on input **x**.

Given program P and input x,

`Halt(P,x)` = { returns `true` if `P(x)` halts
returns `false` otherwise

How might this work?

Clarifications:

`P(x)` is the output of program `P` run on input `x`.

The type of `x` does not matter; assume `string`.

Recap & Next Class

Today:

More lambda calculus

Function graphs

Next class:

Decidability

Reductions