CSCI 334:
Principles of Programming Languages

Lecture 10: Computability

Instructor: Dan Barowy

Williams

1

Topics

Higher order functions

Function graphs

2

Your to-dos

1. Lab 5, **due Monday 03/11** (partner lab)
2. Start studying for the midterm

3

Announcements

- **Midterm exam**, in class, Thursday, March 14.
- Colloquium: **Thinking About Graduate School?** 2:35pm in Wege Auditorium.

CS faculty will discuss your burning questions about graduate school including; deadlines, personal statements, finding an advisor, research, application process, and choosing the right school.

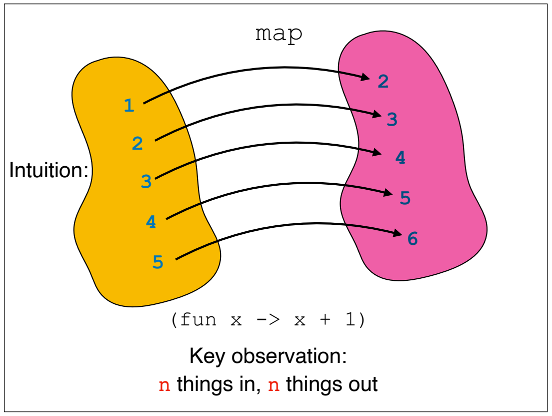
4

Quiz 4

5

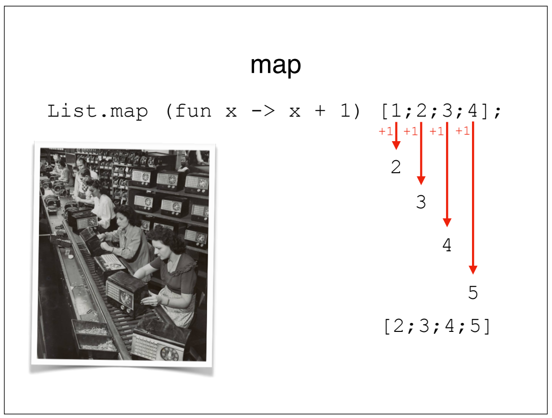
HOF Refresher

6



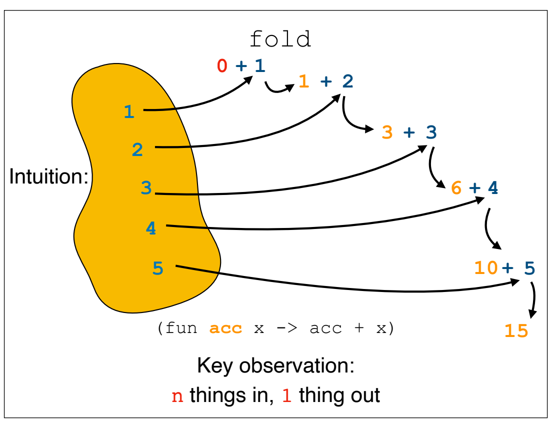
7

map: give it n things, apply function to each one, get n things back.



8

We can use map to add 1 to each element of an input list.



9

fold: give it n things, apply function to each pair of element and accumulator, get 1 thing back.

fold left

```
List.fold (fun acc x -> acc+x) 0 [1;2;3;4]
```

```
acc = 0, [1;2;3;4]
acc = 0+1, [2;3;4]
acc = 1+2, [3;4]
acc = 3+3, [4]
acc 6+4, []
returns acc = 10
```

10 We can use fold to sum all elements of an input list.

---

what does this return?

```
List.fold
  (fun acc x -> acc + string x)
  ""
  (Seq.toList "williams")
```

11 Recall that this returns the list [“w”;”i”;”l”;”l”;”i”;”a”;”m”;”s”]. The input is the list [“w”;”i”;”l”;”l”;”i”;”a”;”m”;”s”] and the initial accumulator is the empty string. For each character, concatenate the character to the accumulator. Return the accumulator.

---

fold right

```
List.foldBack
  (fun x acc -> acc+x) [1;2;3;4] 0
```

```
[1;2;3;4], acc = 0
[1;2;3], acc = 0+4
[1;2], acc = 4+3
[1] acc = 7+2
[], acc = 9+1
returns acc = 10
```

12 You can fold from the left or from the right. Folding right is more expensive on lists than folding left.

what does this return?

```
List.foldBack
  (fun x acc -> acc + string x)
  (Seq.toList "williams")
  ""
```

13 What does this one return?

Activity: folding

```
let number_in_month(ds: Date list)(month: int) : int =
```

- Write a function `number_in_month` that takes a list of dates (where a date is `int*int*int` representing year, month, and day) and an `int month` and returns how many dates are in `month`
- Use `List.fold`

14 Let's try to write a function that uses fold together.

fold

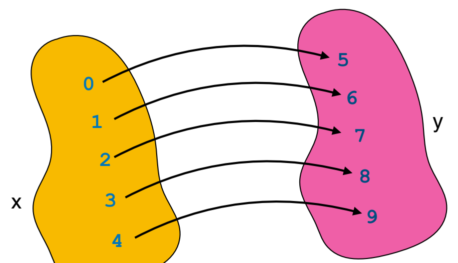
```
let number_in_month(ds: Date list)(month: int) : int =
  ds
  |> List.fold (fun acc (_,mm,_) ->
       if month = mm then
         acc + 1
       else
         acc
     ) 0
```

15 One solution.

Computability

16 Computability is the study of what computers can do, fundamentally.

Intuition: total function

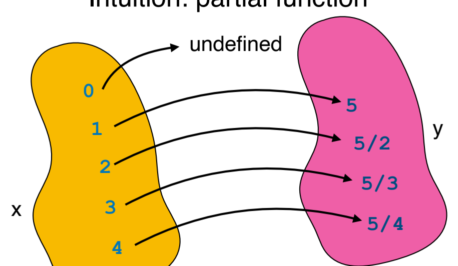


f(x) = x + 5

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

17 First, some terminology. Total function.

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.

18 Partial function.

The **graph** of a function

$f(x) = x + 5$

$\{\langle x, x+5\rangle \mid x \in \mathbb{Z}\}$

$\{\langle x, x+5\rangle \mid x \text{ is an integer}\}$

The graph is **not a picture**!

19 When we want to characterize the behavior of a function, we create what is called a "function graph." This is not a visualization! Instead, we want to state, in terms of set theory, where a function is defined. Specifically, we use set builder notation to state the set of input, output pairs for which the function is defined. Depending on the context, we may restrict ourselves to a domain like integers; we usually do this when discussing computability since discrete (as opposed to continuous) quantities are a computer's natural domain. This graph is for a total function.

The **graph** of a function

$f(x) = 5/x$

$\{\langle x, 5/x\rangle \mid x \in \mathbb{Z} \land x \neq 0\}$

The graph is **not a picture**!

20 Here is a partial function.

Undefinedness

$x/0$

21 Note that we often use "division by zero" as a stand-in for "undefined."

Activity

22

Recap & Next Class

Today:

More HOFs

Function graphs

Next class:

Decidability

Reductions

23