CS131

Saketh Kasibatla

Today

This Class

Ocaml

Homework 1

This Class

Learn and work with 5 different programming languages

Learn about some of the internals of programming languages

Why?

- Make it easier to learn a new language
- Make you a better programmer in the language you already write in
- Help you pick the right language for a new project

Logistics

http://web.cs.ucla.edu/classes/fall17/cs131

sakekasi@ucla.edu (replies from 9am - 8pm)

Office Hours:

MT 9:30-10:30 BH2432

Logistics

http://web.cs.ucla.edu/classes/fall17/cs131/

6 Homework Assignments + 1 Project

1st assignment is due **a week from today**

lateness policy

- 1 day late - 1%
- 2 days late - 2%
- 3 days late - 4%
- and so on

Logistics

Ask all homework/class related questions on Piazza

Submit homework on CCLE

Some homework will be graded using automated scripts

- not compiling → no credit
- your code should behave exactly according to the spec
- function signatures must match, or no credit
- if you are not sure about something, ask for clarification on Piazza
- can talk to your friends about the assignment, but please work alone

OCaml

Install at https://ocaml.org

Or use SEASnet installation

- http://www.seasnet.ucla.edu/lnxsrv/
- use lnxsrv06, lnxsrv07, lnxsrv09
- make sure /usr/local/cs/bin is in your PATH

Setting the PATH

Add the following line to ~/.login

- exec /usr/local/cs/bin/bash

Add the following line to ~/.bash_profile and ~/.bashrc

- create the files if they do not exist
- export PATH=/usr/local/cs/bin:$PATH

To check if the path has been updated correctly, run

- echo $PATH

You should see '/usr/local/cs/bin' at the beginning of the list

Resources

Real World OCaml

- free online at https://realworldocaml.org/

Ocaml Homepage

- https://ocaml.org/learn/

OCaml

static typing → catches lots of programmer errors at compile time

type inference → the system figures out type annotations for you

functional programming → more terse and easier to reason about

used widely in the industry (most notably by Jane Street)

no pointers (at least not in this class)

no objects (at least not in this class)

OCaml

A program is a sequence of bindings

```
let x = 5
let y = x + 1        ← variable bindings
let z = x * y
```

Evaluate the expression to the right in the environment created by the previous bindings

Bind the variable on the left to the value of the expression on the right, and extend the environment

Constants and Operators

```
let a = 2 + 15
val a : int = 17

let b = 49 * 100
val b : int = 4900

let c = (50 * 100) - 4999
val c : int = 1
```

Constants and Operators

```
let d = true
val d : bool = true

let e = false
val e : bool = false

let f true && false
val f : bool = false

let g = not true
val g : bool = false

let h = true || int
ERROR
```

Constants and Operators

```
let i = if 1 < 2 then “a” else “b”
val i : string = “a”

let j = if 1 < 2 then “a” else 2
ERROR
```

Functions

```
let doubleMe x = x + x
val doubleMe : int -> int = <fun>

let k = doubleMe 42
val k : int = 84

let doubleUs x y = x*2 + y*2
val doubleUs : int -> int -> int = <fun>

let l = doubleUs 4 9
val l : int = 26

let doubleOfSum x y =
    let sum = x + y in
    sum * 2
val doubleOfSum : int -> int -> int = <fun>
```

Functions

```
# let max a b = ?
val max : int -> int -> int = <fun>

# let doubleMax a b = ?
val doubleMax : int -> int -> int = <fun>
```

Lists

Same type, variable length

```
let m = [1; 2; 3]
val m : int list = [1; 2; 3]

let n = 1 :: [2; 3] ;;
val n : int list = [1; 2; 3]

let o = 1 :: 2 :: 3 :: [] ;;
val o : int list = [1; 2; 3]

let p = [] ;;
val p : ‘a list = []

let q = [1;2;3]@[4;5;6] ;;
val q : int list = [1;2;3;4;5;6]
```

Tuples

Different types, fixed length

```
let r = (1, 2)
val r : int * int = (1, 2)

let s = (1, “meow”)
val s : int * string = (1, “meow”)

let t = (“a”, 1, “c”)
val t : string * int * string = (“a”, 1, “c”)

let u = fst s
val u : int = 1

let v = snd s
val v : string = “meow”
```

Custom Types

Different branches, each with a tag

```
type maybeString =
| Nothing
| Something of string
type maybeString = Nothing | Something of string

let w = Nothing
val w : maybeString = Nothing

let x = Something “meow”
val x : maybeString = Something “meow”
```

Custom Types

```
type ('nonterminal, 'terminal) symbol =
  | N of 'nonterminal
  | T of 'terminal
type ('nonterminal, 'terminal) symbol = N of 'nonterminal | T of ‘terminal

let y = N “Expr”
val y : (string, ‘a) symbol = N “Expr”

let z = T 42
val z : (‘a, int) = T 42
```

Match

used for case analysis

break down a value by tag

```
let match1 = match [1;2;3] with
  | [] -> “empty”
  | x::xs -> “not empty”
val match1 : string = “not empty”

let match2 = match [2;3;4] with
  | [] -> “empty”
  | _ -> “not empty”
val match2 : string = “not empty”
```

Match

```
let match3 = match (42, 43) with
  | (42, _) -> “42!”
  | (_, 43) -> “43!”
val match3 : string = “42!”

let match4 = match T “3” with
  | N _ -> “nonterminal”
  | T x -> “terminal” + x
val match4 : string = “terminal 3”
```

Functions + Lists

```
type ‘a option =
| None
| Some of ‘a

let head list = ?
val head : ‘a list -> ‘a option = <fun>

let tail list = ?
val tail : ‘a list -> ‘a list = <fun>
```

Functions + Lists

```
let rec length list = match list with
  | [] -> ?
  | x::xs -> ?
val length : ‘a list -> int = <fun>

let fn = (fun x -> x + 1)
val fn : int -> int = <fun>

let ans = (fun x -> x + 1) 2
val ans : int = 3

let rec map_helper fn list agg = match list with
  | [] -> ?
  | x::xs -> ?
let map fn list = map_helper fn list []

let ans2 = map (fun x -> x + 1) [1;2;3]
val ans2 : int list = [2;3;4]
```

Functions + Lists

```
let insertAfterFirst list item = ?
val insertAfterFirst : ‘a list -> ‘a -> ‘a list = <fun>

let ans3 = insertAfterFirst [1;3;4] 2
val ans3 : int list = [1;2;3;4]

let rec fold list fn initial = match list with
  | [] -> ?
  | x::xs -> ?
val fold : 'a list -> ('b -> 'a -> 'b) -> 'b -> 'b = <fun>

let ans4 = fold [1;2;3] (fun x y -> x + y) 0
val ans4 : int = 6
```

Fixpoints

when you call a function over and over, does it stay at some number?

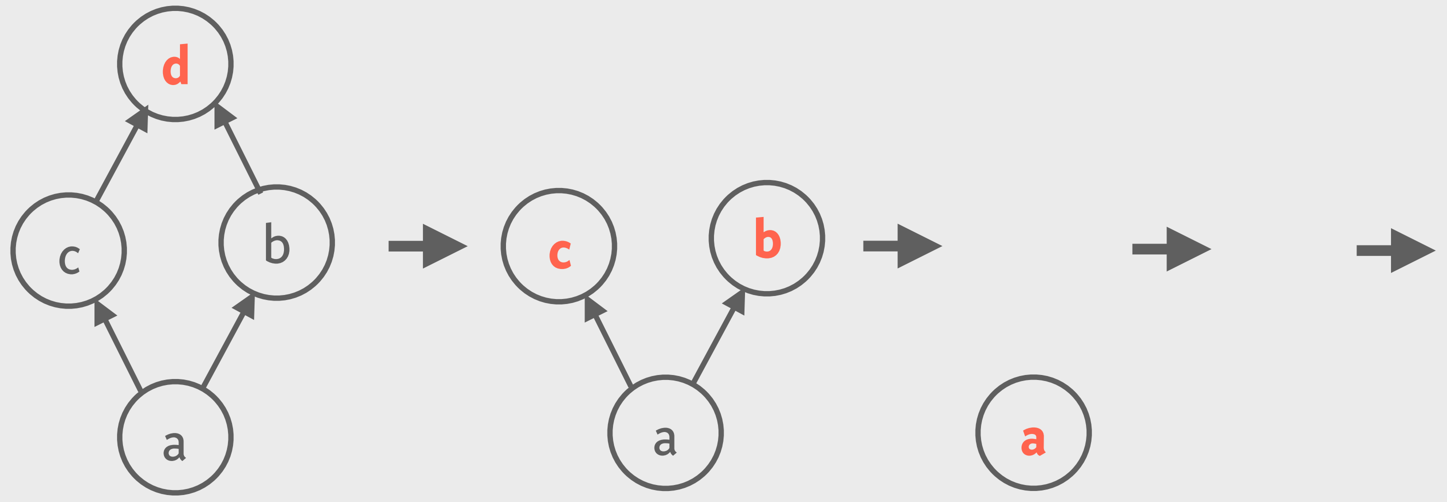
in other words, for what x does f(x) = x?

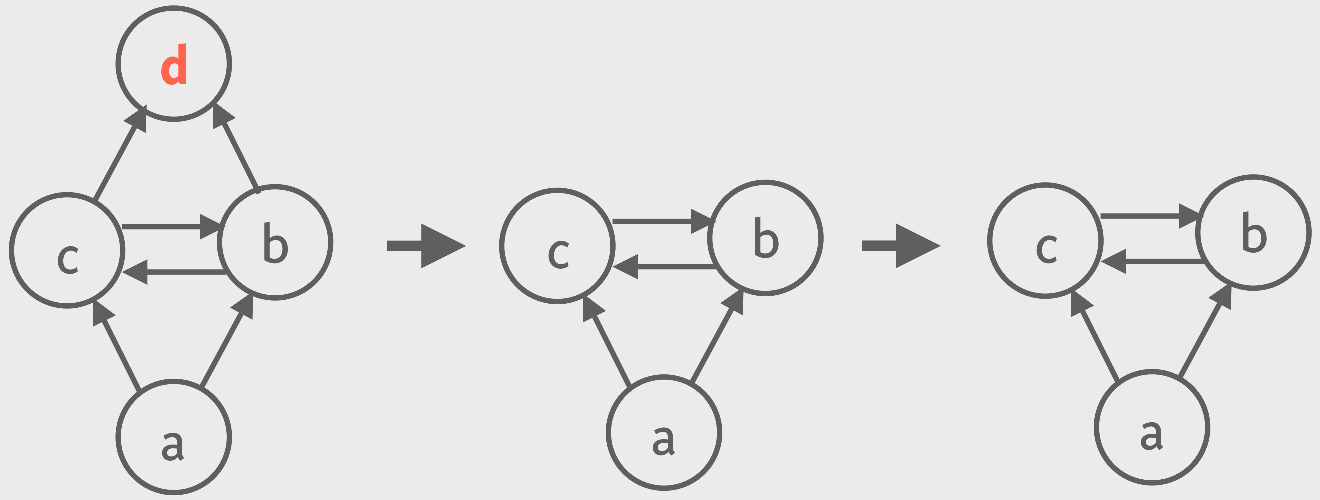
```
let f x = x / 2

   f     f     f     f     f     f
10 →  5  →  2  →  1  →  0  →  0  →  0
```

Fixpoints

dependency resolution

function to repeat: take everything without a dependency and remove it





Grammars

a way to describe which strings are and aren't valid for a language

symbol

- terminal - a symbol that can't be replaced with other symbols
- nonterminal - a symbol that can be replaced with other symbols

rule - a list of symbols that can replace a nonterminal symbol

grammar - a starting symbol and a set of rules that describe which symbols can be derived from a nonterminal

Grammars

Symbols: ? Nonterminals: ? Terminals: ? Starting symbol: S

S → A

S → B

A → aA

B → bB

B → b

How to derive 'Aaa'?

Grammars

Symbols: S, A, B, a, b   Nonterminals: S, A, B   Terminals: a, b Starting symbol: S

rules:

S → A

S → B

A → aA

B → bB

B → b

How to derive 'aaa'?

1. S
2. A (S → A)
3. aA (A → aA)
4. aaA (A → aA)
5. aaa (A → a)

Blind Alleys

Rules from which it is impossible to derive a string of terminals

Symbols: S, A, B, a, b Nonterminals: S, A, B Terminals: a, b Starting symbol: S

Rules:

S → A

S → B

A → A ⟵ **What are the blind alley rules?**

A → aB

A → aA

A → a

B → B