Andrew Appel David Walker

COS 326 Functional Programming: An elegant weapon for a more civilized age

Princeton University

Your professors

Andrew Appel

David Walker

Your professors

Andrew Appel

David Walker

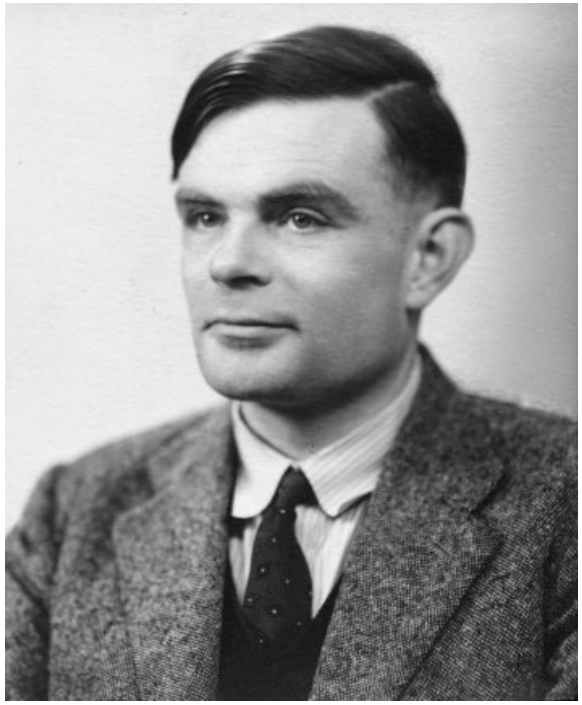
Alonzo Church, 1903-1995
Princeton Professor, 1929-1967

In 1936, Alonzo Church invented the lambda calculus. He called it a logic, but it was a language of pure functions -- the world's first programming language.

He said:

"*There may, indeed, be other applications of the system than its use as a logic.*"

Indeed!



Alonzo Church
1934 -- developed lambda calculus

Programming Languages

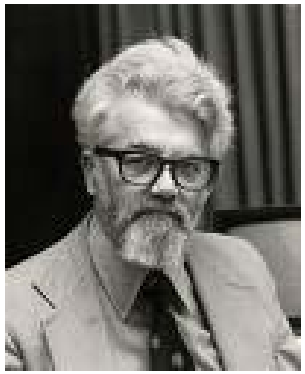
Alan Turing (PhD Princeton 1938)
1936 -- developed Turing machines

Computers

Optional reading: ***The Birth of Computer Science at Princeton in the 1930s***
by Andrew W. Appel, 2012. http://press.princeton.edu/chapters/s9780.pdf

A few designers of functional programming languages

Alonzo Church:
λ-calculus, 1934

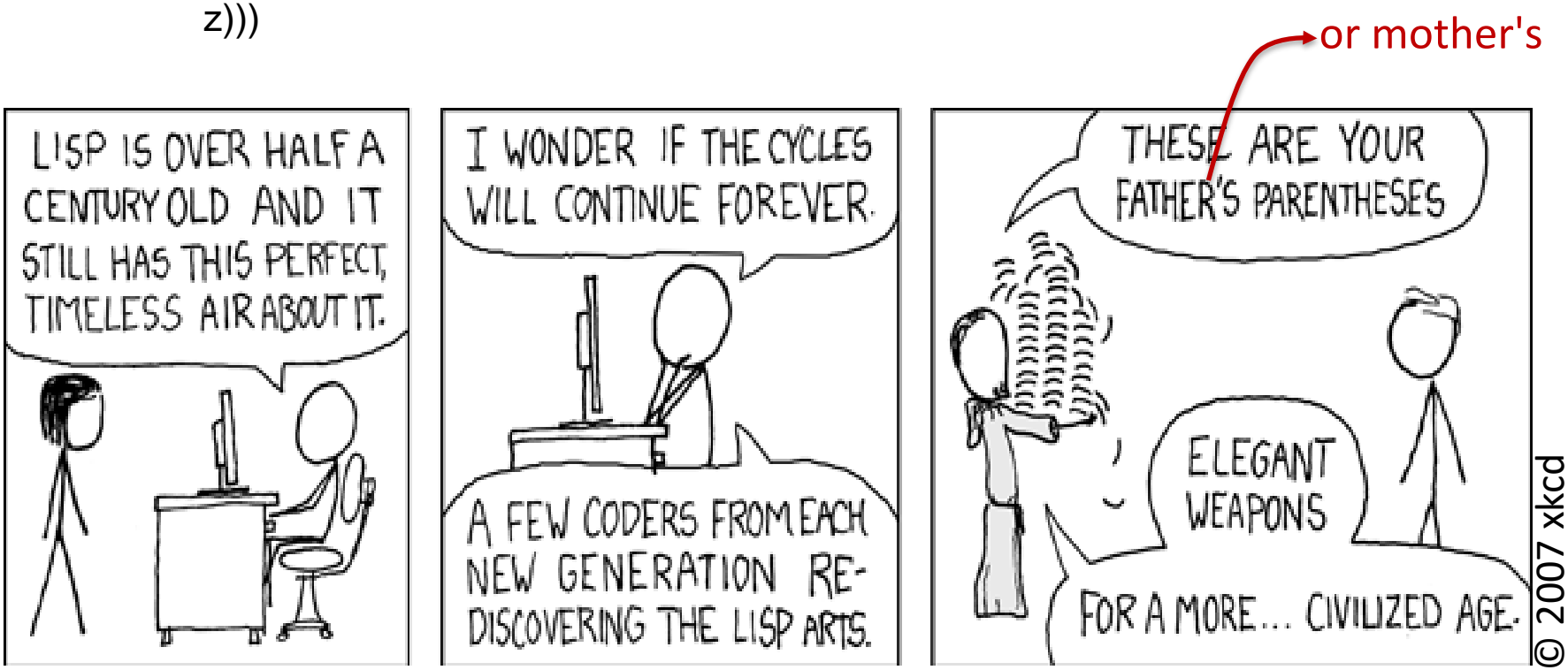


John McCarthy
(PhD Princeton 1951)
LISP, 1958

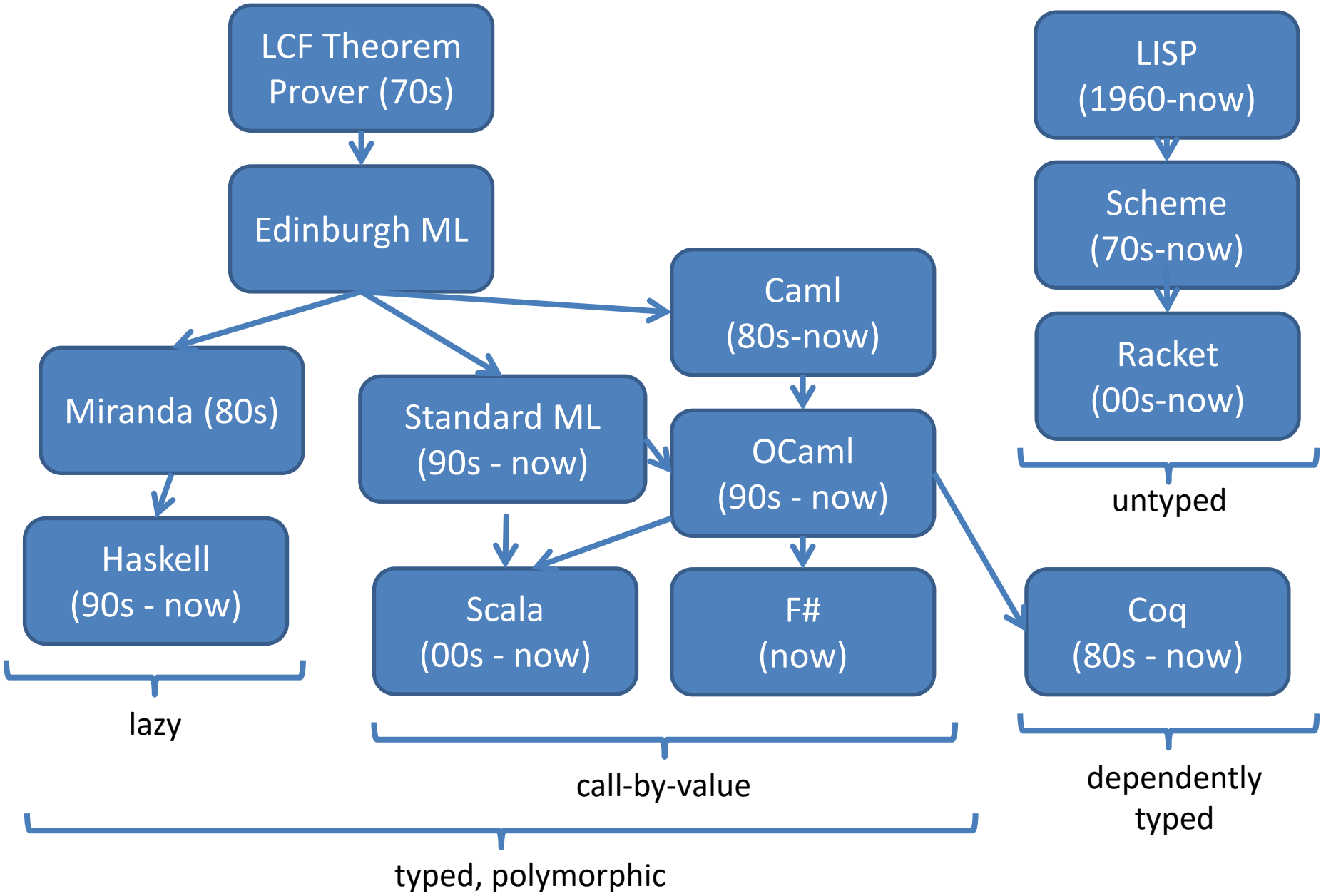
Guy Steele & Gerry Sussman:
Scheme, 1975

LISP, 1960

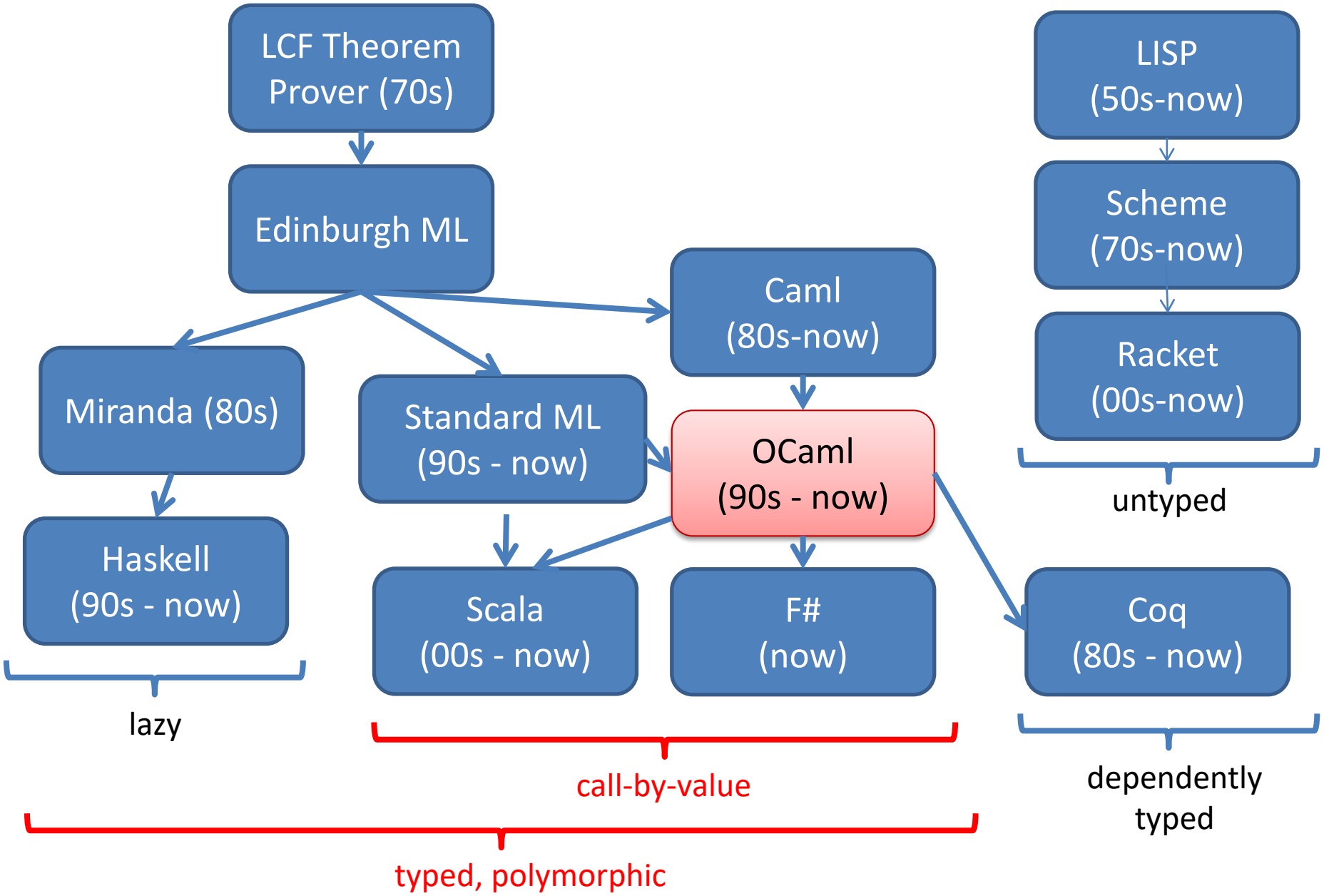
```
(define
    (my-max3 x y z)
    (if (and (> x y) (> x z))
        x
        (if (> y z)
            y
            z)))
```



Vastly Abbreviated FP Genealogy



Vastly Abbreviated FP Geneology



Functional Languages: Who's using them?

map-reduce in their data centers

Scala for
correctness, maintainability, flexibility

Erlang for concurrency,
Haskell for managing PHP,
OCaml for bug-finding

F# in Visual Studio

mathematicians

Coq (re)proof of
4-color theorem

Haskell to
synthesize hardware

Haskell
for specifying
equity derivatives

www.artima.com/scalazine/articles/twitter_on_scala.html
www.infoq.com/presentations/haskell-barclays
www.janestreet.com/technology/index.html#work-functionally
msdn.microsoft.com/en-us/fsharp/cc742182
research.google.com/archive/mapreduce-osdi04.pdf
www.lightbend.com/case-studies/how-apache-spark-scala-and-functional-programming-made-hard-problems-easy-at-barclays
www.haskell.org/haskellwiki/Haskell_in_industry

COURSE LOGISTICS

Course Staff

Professors

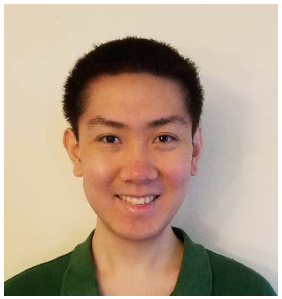
Andrew Appel

David Walker

Preceptors

Akash Gaonkar

Joomy Korkut



John Li

Devon Loehr

Danqi Liao

Dmitry Paramonov

Resources

- coursehome: http://www.cs.princeton.edu/~cos326
- Communication
  - Ed: https://us.edstem.org/courses/2192/discussion/
- Lecture schedule and readings:
  - $(coursehome)/lectures.php for schedule and slides
  - Lectures this semester will be prerecorded videos
  - Join the professors for (optional) “lecture watching party”
    - Tues-Thurs 11am; ask questions of your profs during lecture!
    - Zoom link on Ed; authenticate with your princeton.edu login
- Assignments:
  - $(coursehome)/assignments.php
- Precepts: (one hour per week)
  - Precept attendance is mandatory.
- Install OCaml: $(coursehome)/resources.php

A Typical Week

Monday

- Assignment from last week due (7 assignments total)
- Your first assignment is due Monday Sept 7 at 11:59pm

Tuesday

- New assignment handed out
- video watch party for lectures at 11-12:20 on zoom (see ed)
  - profs present to answer questions
- *start assignment* with material from lecture

Thursday

- video watch party 11-12:20

Thursday/Friday

- mandatory precept reinforces lecture content in small groups
- you may have questions for your preceptor about the assignment

Collaboration Policy

The COS 326 collaboration policy can be found here:

http://www.cs.princeton.edu/~cos326/info.php#collab

Read it in full prior to beginning the first assignment.

Please ask questions whenever anything is unclear, at any time during the course.

Sample README.txt (abridged)

Netids (include all members of group, if parternering):
[list here]

In doing this homework I used the following sources:

1. Sources I don't need to mention [see notes 1 and 4]
2. Authorized sources [see notes 2 and 4]
   [list here]
3. Unauthorized sources [see notes 3 and 4]
   [list here]

This paper represents my own work in accordance with University regulations.

Signed, [your name(s):]

NOTE 1: Sources you don't need to mention

this semester's lectures and precepts, the course web site, the assignment handout (download), Real World OCaml, and the OCaml manual.

NOTE 2: Authorized sources include:

professors and preceptors, advice from other students (but not looking at their solutions); other books, and (within reason) web sites such as stackoverflow.com.

NOTE 3: "Why would I list an unauthorized source?"

Using an unauthorized source without citing it is an Academic Violation under Princeton University's disciplinary code, and can result in suspension from the University.

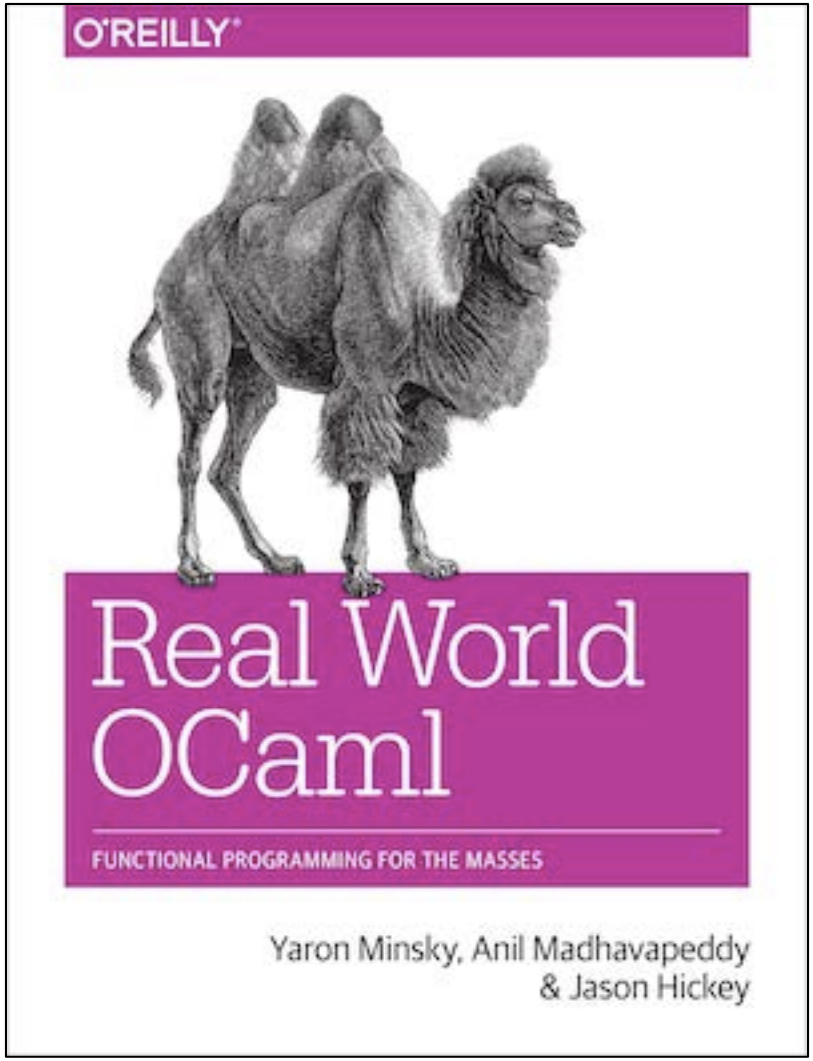
Using an unauthorized source and citing it clearly is "merely" a violation of this homework's instructions, and can result in (at most) getting a zero on this homework.

Unauthorized sources include, at least: other people's solutions to these (or similar) homework problems.

NOTE 4. If you paste in code from from these sites, you should clearly cite it at the point of use, in accordance with Section 2.4.6 of RRR for "direct quotation or extensive paraphrase".

Please limit the amount of this that you do in accordance with the principle that the purpose of these homeworks is so that you can learn how to do things yourself.

Course Textbook



http://realworldocaml.org/

Exams

Midterm

- Thursday after your (Sat→Tues) midterm break

Final

- During exam period in December
- The final is ***not*** "cumulative" over the whole semester, it covers just "equational reasoning"

Assignment 0

Download and install OCaml and get emacs or your favorite editor set up to process OCaml code (syntax highlighting; type checking)

See the Resources Page for install instructions on your platform:

http://www.cs.princeton.edu/~cos326/resources.php

Thinking Functionally

pure, functional code:

```
let (x,y) = pair in
(y,x)
```

you *analyze* existing data (like pair) and you *produce* new data (y,x)

imperative code:

```
temp = pair.x;
pair.x = pair.y;
pair.y = temp;
```

commands *modify* or *change* an existing data structure (like pair)

Thinking Functionally

pure, functional code:

```
let (x,y) = pair in
(y,x)
```

- *outputs are everything!*
- *output is function of input*
- *data properties are stable*
- *repeatable*
- *parallelism apparent*
- *easier to test*
- *easier to compose*

imperative code:

```
temp = pair.x;
pair.x = pair.y;
pair.y = temp;
```

- *outputs are irrelevant!*
- *output is not function of input*
- *data properties change*
- *unrepeatable*
- *parallelism hidden*
- *harder to test*
- *harder to compose*

This simple switch in perspective can change the way you *think* about programming and problem solving.

Have fun!

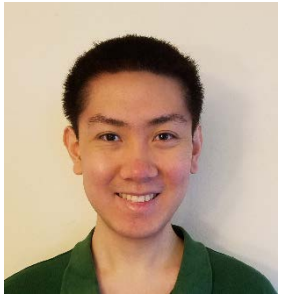
Let's make this an amazing semester!

Andrew Appel

David Walker

Akash Gaonkar

Joomy Korkut



John Li

Devon Loehr

Danqi Liao

Dmitry Paramonov