CSE 130 — Programming Languages

Notes taken by Nolan Chai

Spring 2023

Contents

Preface

These are a collection of notes personally taken by me, specifically for readings and allotted content for UCSD's CSE 130 Programming Languages, taken in Spring 2023. These notes are not endorsed by the lecturers nor staff, and I have modified them (often significantly) over random periods of time. They may become nowhere near accurate representations of what was actually lectured, or written in the books, and are simply to aid in my own understanding. In particular, all errors are almost surely mine.
Notes are taken real time, and will be reviewed, updated, and revised within 48 hours of each lecture.

My other notes are available **here**.

0 Introduction

Programming Languages are the duct tape, bricks, mortar and steel of the information age. Over the last thirty years, a variety of languages with diverse features have been developed, expressing radically different perspectives of the idea of computation. CSE 130 is an introduction to some of these perspectives, as well as the fundamental concepts of languages. We shall focus in particular on one paradigm – functional programming as embodied in the lambda calculus and the Haskell language. Many students will be encountering this paradigm and these languages for the first time. As with spoken languages, these are best absorbed by immersing yourself in the different environments and practicing your skills by experimentation. (Courtesy of the course website)

Logistics

There will be **one midterm** on **Monday, May 8th, 2023**, in the format of a multiple choice quiz on Gradescope open for 24 hours. The **final exam** will be held on **Thursday, June 15** in the format of a programming assignment to be solved in 24 hours. Since the final is cumulative, your midterm grade will be calculated as

```
midterm > 0 ? max(final, midterm) : 0
```

This means that you get a second chance if you don't do well on the midterm but you must turn in both the midterm and the final.

There are a total of **six programming assignments**, assigned once every week or two. You can work on them in groups of **up to two people.** You have a total of **8 late days**, but no more than **4 late days** per assignment.

- PA0 - Due 4/19
- PA1 - Due 4/26
- PA2 - Due 5/5
- PA3 - Due 5/12
- PA4 - Due 5/24
- PA5 - Due 6/7

Lectures will be hybrid - both in-person and live on zoom + recorded.

Grading Scheme

- 45% Homework assignments
- 25% Midterm
- 30% Final
- 5% Extra credit for Piazza discussion
  - To top 20 best participants

1 Hello World!

Some related research Professor Nadia Polikarpova does work on that you may want to contact if interested: Program verification and synthesis - how to prove the program is doing the right thing, and how to generate a program to do the right thing.

1.1 A Programming Language

The work in this class will primarily revolve around the functional language Haskell. Given two variables - x, y, and the commands

- $x++$
- $x--$
- $x == 0 \ ? \ L1 \ : \ L2$

Given the following script, how would we describe this?
Note that $x == 0 \ ? \ L1 \ : \ L2$ *is a ternary operator that marks if x=0 then jump to the label L1, else jump to the label L2.*

```
L1: x++
    y--
    y = 0 ? L2 : L1
L2: ...
```

This computes the sum of x and y. It turns out that the above language, in some sense, is "equivalent" to every programming language - it has the computational power of all other PLs! However, good luck writing the following...

- QuickSort
- Fortnite
- Spotify

Two main takeaways you should have from this class:

- All Programming Languages share something fundamental
- But also: your Programming Language shapes your Programming Thought

We aren't specifically learning languages, but rather, learning a fundamental understanding of PLs and computational thinking to design and implement languages. There are two schools of thoughts in teaching PLs - interpreters (several variations in a language) vs (a "zoo"-style course) teaching multiple languages and noting their differences. This format will follow the interpreter 'school of thought'. A good language wants to do its best to be:

- Correct
- Readable
- Extendable
- Reusable

1.2 Course Schedule

Here is a course outline with some brief descriptions and questions to be covered per topic:

Lambda Calculus (2 weeks)

- The simplest language on Earth

Haskell (5 weeks)

- A cool functional language

Build Your Own Language

- How do we implement a new language (Haskell)?
- How do we formalize a language and prove things about it?

1.3 Quicksort in C v.s. Haskell

```c
void sort(int arr[], int beg, int end){
  if (end > beg + 1){
    int piv = arr[beg];
    int l = beg + 1;
    int r = end;
    while (l != r-1)
       if(arr[l] <= piv) l++;
       else swap(&arr[l], &arr[r--]);
    if(arr[l]<=piv && arr[r]<=piv)
       l=r+1;
    else if(arr[l]<=piv && arr[r]>piv)
       {l++; r--;}
    else if (arr[l]>piv && arr[r]<=piv)
       swap(&arr[l++], &arr[r--]);
    else r=l-1;
    swap(&arr[r--], &arr[beg]);
    sort(arr, beg, r);
    sort(arr, l, end);
  }
}
```

Versus only **4 lines in Haskell!**

```haskell
sort []     = []
sort (x:xs) = sort ls ++ [x] ++ sort rs
  where
    ls      = [ l | l <- xs, l <= x ]
    rs      = [ r | r <- xs, x <  r ]
```

Not the most fair comparison, but you can see how powerful functional languages can be.

2 The Lambda Calculus