

CPSC-354 Report

Michael Masakayan
Chapman University

October 10, 2022

Abstract

Short summary of purpose and content.

Contents

1	Introduction	1
2	Homework	1
2.1	Week 1	1
2.2	Week 2	2
2.3	Week 3	4
2.4	Week 4	4
3	Project	8
3.1	Project Outline	8
4	Conclusions	8

1 Introduction

1 My name is Michael Masakayan, I'm a 4th year computer science major. I transferred from Pasadena City College in 2020. I am interested in cyber security and although I won't be able to get a minor in it I plan on getting a few certificates this year. I enjoy playing video games and like watching movies

2 Homework

This section will contain your solutions to homework.

2.1 Week 1

Assignment 1: GCD. The program figures out the Greatest Common Divisor for any given two numbers using Euclid's algorithm.

1. Firstly you want to put the program into your IDE and run `g++ -o assignment1 assignment.cpp`
2. then type in `./assignment1.cpp`
3. Once you run the program you type in the first number then you hit enter then you type in the second number and hit enter. It will then show you the GCD of the two numbers.

I chose to use c++ because I am familiar with the language and it was easier for me to write the program in it. This assignment was mainly for us to get used to using LaTeX

```
#include <iostream>
using namespace std;
int gcd(int n, int m) {
    if (m == 0)
    {
        return n;
    }

    else if(n>m)
    {
        gcd(m,n-m);
    }
    else if(n<m)
    {
        gcd(n,m-n);
    }
    return gcd(m, n % m);
}
int main() {
    int n,m;
    cout<<"type the number for the first argument of the GCD, then hit enter"<<'\\n';
    cin>>n;
    cout<<"type the number for the first argument of the GCD,then hit enter"<<'\\n';
    cin>>m;
    cout<<"GCD of "<< n <<" and "<< m <<" is "<< gcd(n, m);
    return 0;
}
```

2.2 Week 2

Assignment 2: In this assignment we were tasked with creating a few recursive functions in haskell that is

1. select_{evens} select_{dds}
2. member
3. append
4. revert
5. less_{equal}

Select evens will take a list and return a another list with only the even elements of the list in the argument.

select_{dds} is meant to do what select evens does but for the odd items in the list. Member, is meant to check a list if it has the given argument

6. Firstly you want to put the program into your IDE and run ghci

7. then type in :load assignment2.hs

8. you can type the following and they should give the results

(a) $\text{select}_{evens}["e", "f", "g", "h", "i"] - \text{result}["f", "h"]$

(b) $\text{select}_{dds}["e", "f", "g", "h", "i"] - \text{result}["f", "h", "i"]$

- (c) member 3 [2,4,3]—result true member 3 [2,4,2]—result false
- (d) append [2,1] [3,1,2] —result [2,3,1,1,2]
- (e) revert [5,6,7,8] —result [8,7,6,5]
- (f) less_equal[1][2] — *—resulttrueless_equal[3][2] — —resultfalse*

I mainly had issue with less equal. I could not find the syntax for how to recursively go through more than one element lists. Most of them used the same type of recursion.

```

{-let eexample_eveny = ["a","b","c","d","e"] -}
{-yselect_eveny (z:zy) n = z !!n if z !! n/2 == % 0 then b:a!! n
-}
select_evens:: [a] -> [a]
select_evens [] = []
select_evens [x] = []
select_evens (x1:x2:xy) = x2:select_evens xy
-- selects all the odd elements in a list
select_odds :: [a] -> [a]
select_odds [] = []
select_odds [x] = [x]
select_odds (x1:x2:xy) = x2:select_odds xy
-- member takes an int and a list and checks if the int is in the list

member:: Eq a=> a-> [a] -> Bool
member _ [] = False
member x (z:zs)= (x `elem` zs) == True

-- combines two lists into one list
append :: [a] -> [a] -> [a]
append [] xy = xy
append (z:zy) xy = z:append xy zy
-- reverses the order of a list
revert :: [a] -> [a]
revert xs = rev [] xs where
    rev :: [a] -> [a] -> [a]
    rev acc [] = acc
    rev acc (x:xs) = rev (x:acc) xs
--compares the values of the lists
less_equal:: Ord a=> [a] -> [a] -> Bool
less_equal [] [] = True
less_equal [x][b] = (x>=b)
-- less_equal (x:xs) (b:bz) = x<b:less_equal xs bz

main::IO ()
main = print(append [1,2] [3,4,5])
-- print(less_equal [] [])
-- print(select_odds ["a","b","c","d","e"])
-- print (revert [1,2,3])

```

2.3 Week 3

For this week we went over Parsing and Context-Free Grammars. Question 1: write out the derivation trees (also called parse trees or concrete syntax trees) for the following strings:

- (g) $2+1$
- (b) $1+2*3$
- (c) $1+(2*3)$
- (d) $(1+2)*3$
- (e) $1+2*3+4*5+6$
- (f)

Question 1.1: Is the abstract syntax tree of $1+2+3$ identical to the one of $(1+2)+3$ or the one of $1+(2+3)$?
Answer: No the abstract syntax tree is not identical to either of them you need to go through to get the paranthesis this is not completely the case when dealing with the derivation tree though.

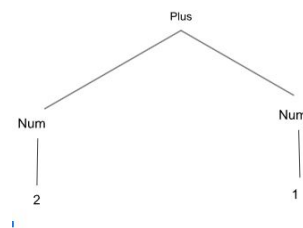


Figure 1: $2+1$ derivation tree

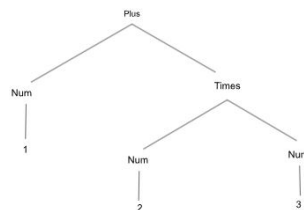


Figure 2: $1+2*3$ derivation tree

Question 1.2: write out the abstract syntax trees for the following strings:

2.4 Week 4

For homework 6 we were suppose to evaluate:

- $(\lambda exp.\lambda two.\lambda three.exp\ two\ three)$
- $(\lambda m.\lambda n.mn)$

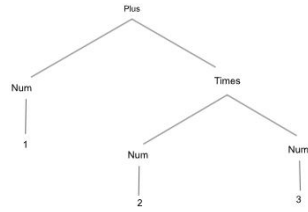


Figure 3: $1+(2*3)$ derivation tree

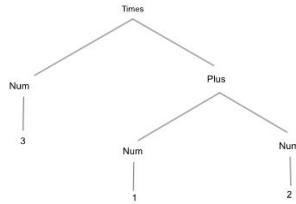


Figure 4: $(1+2)*3$ derivation tree

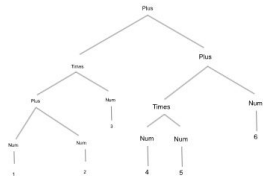


Figure 5: $1+2*3+4*5+6$ derivation tree

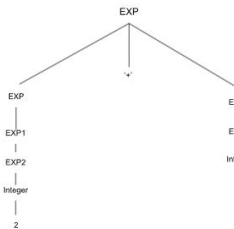


Figure 6: $2+1$ abstract syntax tree

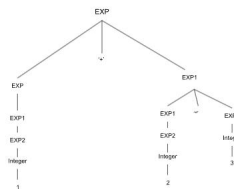


Figure 7: $1+2*3$ abstract syntax tree

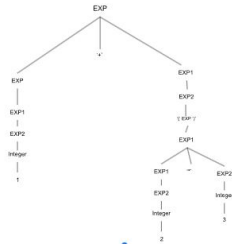


Figure 8: $1+(2*3)$ abstract syntax tree

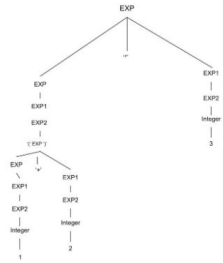


Figure 9: $(1+2)*3$ abstract syntax tree

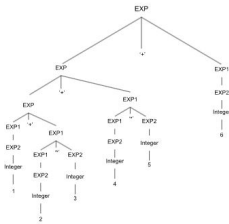



Figure 10: $1+2*3+4*5+6$ abstract syntax tree


- $(\lambda f.\lambda x.f(fx))$
- $(\lambda f.\lambda x.f(f(fx)))$


From here we did a step by step simplifying the problem. We went over this class and there was a video teaching us on how we should do it. We also were suppose to go over our project outline and solitify what we will be doing for the final project


- $=((\lambda m.\lambda n.mn)(\lambda f.\lambda x.f(fx))(\lambda f.\lambda x.f(f(fx))))$
- $=((\lambda m.\lambda n.mn)(\lambda f.\lambda x.f(fx))(\lambda f2.\lambda x2.f2(f2(f2x2))))$
- $=((\lambda f.\lambda x.f(fx))(\lambda f2.\lambda x2.f2(f2(f2x2))))$
- $=((\lambda x.(\lambda f2.\lambda x2.f2(f2(f2x2))((\lambda f2.\lambda x2.f2(f2(f2x2))x))))$
- $=((\lambda x.(\lambda f2.\lambda x2.f2(f2(f2x2))((\lambda x2.x(x(x2))))$
- $=((\lambda x.\lambda x2.\lambda x2.x(x(x2))((\lambda x2.x(x(x2))(\lambda x2.x(x(x2))x2))))$
- $=((\lambda x.\lambda x2.\lambda x2.x(x(x2))(\lambda x2.x(x(x2))(x(x(x2))))))$
- $=((\lambda x.\lambda x2.\lambda x2.x(x(x2))(x(x(x(x(x2))))))$
- $=((\lambda x.\lambda x2.x(x(x(x(x(x(x2))))))))$


We also had to do the excersices we had to do in class.I have attached the photo in latex and in the github if it doesn't load it should be hw6.jpg


confluent \vee
 ϵ -reducing x
 λ . normal forms x 


confluent \vee
 ϵ -reducing x
 λ . normal forms \vee 

confluent \vee
 ϵ -reducing \vee
 λ . normal forms x 

confluent x
 ϵ -reducing \vee
 λ . normal forms \vee 

confluent x
 ϵ -reducing \vee
 λ . normal forms x 

confluent x
 ϵ -reducing x
 λ . normal forms \vee 

confluent x
 ϵ -reducing x
 λ . normal forms x 

3 Project

3.1 Project Outline

I was looking into having my project being more code and programming based since I am graduating soon and would like to have a project to put on my resume. I was really interested in making a webscrapping tool using haskell and scalpel but I think that might be a little too ambitious. But in terms of coding projects I am really interested in visual code like p5.js and processing. If there are any projects that involve creative and visual programming with haskell please let me know. Something like Perlin noise fields, 10PRINT, or even doing some ASCII art. But until I find something that I will be able to do like that I think I am going to try and make a program that does Mandelbrot sets in ASCII. I think that this is a achievable program that I could make in haskell. So I will be hopefully making graphical depiction of the Mandelbrot set. In the project I will be going over what Mandelbrot sets and how they can be graphically shown. Then I will be going into how it specifically works within haskell.

4 Conclusions

(approx 400 words)

In the conclusion, I want a critical reflection on the content of the course. Step back from the technical details. How does the course fit into the wider world of programming languages and software engineering?

References

[PL] [Programming Languages 2022](#), Chapman University, 2022.