CPSC-354 Report

Michael Masakayan
Chapman University

November 7, 2022

Abstract

Short summary of purpose and content.

Contents

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

```cpp
#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 recursiver functions in haskel that is

1. $select_e vens select_o dds$

2. member

3. append

4. revert

5. $less_e qual$

Select evens will take a list and return a another list with only the even elements of the list in the argument. $select_o dds is meant to do what select even does but for the odds items in the list. Member, is meant to check a list if it has the given argum$

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) $select_evens["e", "f", "g", "h", "i"] -- result["f", "h"]$

(b) $select_odds["e", "f", "g", "h", "i"] -- result["f", "h", "i"]$

(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) $less_equal[1][2] -- result\ true\ less_equal[3][2] -- result\ false$

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.

```haskell
{-let ezample_eveny = ["a","b","c","d","e"] -}
{-yelect_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 6

For homework 6 we were suppose to evaluate:

(g) ($\lambda exp.\lambda two.\lambda three.exp two three$)

- ($\lambda m.\lambda n.mn$)
- ($\lambda f.\lambda x.f(fx)$)
  - ($\lambda f.\lambda x.f(f(fx))$)

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

  - $=((\lambda m.\lambda n.mn)(\lambda f.\lambda x.f(fx))(\lambda f.\lambda x.f(f(fx))))$
- $=((\lambda m.\lambda n.mn)(\lambda f.\lambda x.f(fx))(\lambda f2.\lambda x2.f2(f2(f2x2))))$
- $= (((\lambda f\lambda x.f(fx))(\lambda f2.\lambda x2.f2(f2(f2x2)))))$
- $=(((\lambda x.(\lambda f2.\lambda x2.f2(f2(f2x2)((\lambda f2.\lambda x2.f2(f2(f2x2)x)))$
- $=(((\lambda x.(\lambda f2.\lambda x2.f2(f2(f2x2)((\lambda x2.x(x(xx2))))$
- $=(((\lambda x.\lambda x2.\lambda x2.x(x(xx2))((\lambda x2.x(x(xx2))(\lambda x2.x(x(xx2))x2))))$
- $=(((\lambda x.\lambda x2.\lambda x2.x(x(xx2))(\lambda x2.x(x(xx2))(x(x(xx2))))))$
- $=(((\lambda x.\lambda x2.\lambda x2.x(x(xx2))(x(x(x(x(x(xx2)))))))$
- $=(((\lambda x.\lambda x2.x(x(x(x(x(x(x(x(xx2))))))))))$
  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 dosn't load it should be hw6.jpg

confluent ∨
terminating x
u. normal forms x

confluent ∨
terminating x
u. normal forms ∨

confluent ∨
terminating ∨
u. normal forms x

confluent x
terminating ∨
u. normal forms ∨

confluent x
terminating ∨
u. normal forms x

confluent x
terminating x
u. normal forms ∨

confluent x
terminating x
u. normal forms x

2.4   Week 7

2) If you have not done the evalCBN part of hw5, do it for this homework.

```
'(\x.\y.x) y z'

evalCBN ((EApp (EAbs (Id "x") (EVar (Id "x"))) (EApp (EAbs (Id "y") (EVar (Id "y"))) (EVar (Id
    "a")))))
== line 6
evalCBN (subst (Id "x" ) (EApp (EAbs (Id "y") (EVar (Id "y"))) (EVar (Id "a"))) (Evar (Id "x")))
== line 15
evalCBN (EApp (EAbs (Id "y") (EVar (Id "y"))) (EVar (Id "a")))
== line 6
evalCBN (subst (Id "y") (EVar (Id "a")) (EVar (Id "y")))
 == line 15
evalCBN (EVar ( Id "a"))
 == line 8
EVar (Id "a")

evalCBN (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "x")))) (EVar (Id "y")) (EVar (Id "z")))
== line 6
evalCBN(subst (Id "x") (EAbs (Id "y") (EVar (Id "x"))) (EVar (Id "y")) (EVar (Id "z")))
 == line 15
```

5

```
evalCBN(EApp (EAbs (Id "y") (EVar (Id "y"))) (EVar (Id "z")))
== line 6
evalCBN (subst (Id "y") (EVar (Id "z")) (EVar (Id "y")))
 == line 15
evalCBN (EVar (Id "z"))
 == line 8
EVar (Id "z")
```

---

confluent   ∨
terminating   x
u. normal forms x

confluent   ∨
terminating   x
u. normal forms ∨

confluent   ∨
terminating   ∨
u. normal forms x

confluent   x
terminating   ∨
u. normal forms ∨

confluent   x
terminating   ∨
u. normal forms x

confluent   x
terminating   x
u. normal forms ∨

confluent   x
terminating   x
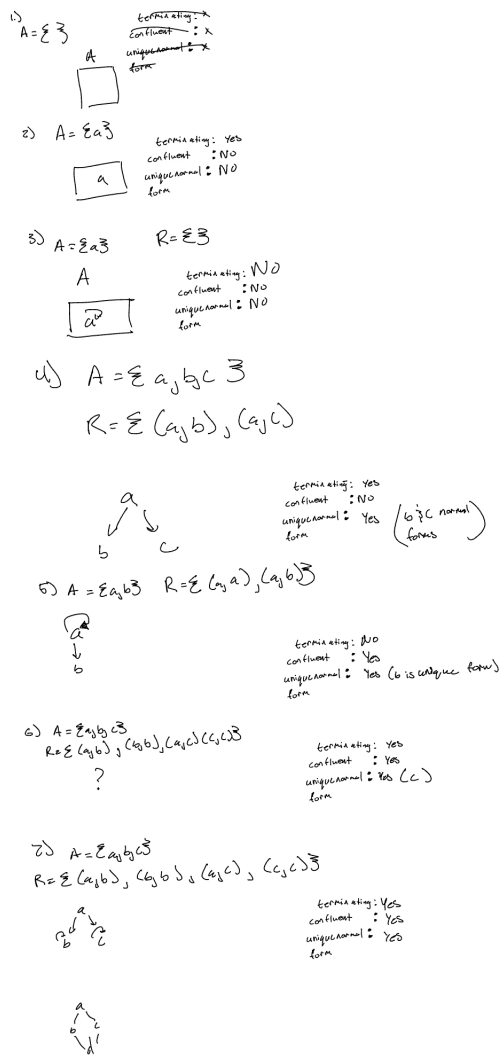u. normal forms x

1) $A = \{\}$ terminating: x   confluent: x   unique normal form: x

2) $A = \{a\}$ terminating: Yes   confluent: No   unique normal: No

3) $A = \{a\}$ $R = \{\}$ terminating: No   confluent: No   unique normal form: No

4) $A = \{a, b, c\}$ $R = \{(a, b), (a, c)\}$ terminating: Yes   confluent: No   unique normal form: Yes (b, c normal forms)

5) $A = \{a, b\}$ $R = \{(a, a), (a, b)\}$ terminating: No   confluent: Yes   unique normal: Yes (b is unique form)

6) $A = \{a, b, c\}$ $R = \{(a, b), (a, b), (a, c), (c, c)\}$ terminating: Yes   confluent: Yes   unique normal form: Yes (c)

7) $A = \{a, b, c\}$ $R = \{(a, b), (c, b), (a, c), (c, c)\}$ terminating: Yes   confluent: Yes   unique normal form: Yes

2.5 Week 8

In week 8 we went over string rewriting exercises. We were shown how to analyze the ARS for each of given an algorithm by the professor.

1. Q) Why does the ARS not terminate? A)The Ars does not terminate because the rules will reduce the left side lengths so it will end up reduced.

2. Q) What are the normal forms? A)The normal forms would be "a, b" besides that I do not think there are normal forms unless I am misunderstanding.

3. Q)Can you change the rules so that the new ARS has unique normal forms (but still has the same equivalence relation)? A)Yes you can change the rules so that the new ARS has unique forms I believe we went over this class when we did and rewrote.

```
b b-> a
ab ->b
```

```
ba ->b
```

4. Q) What do the normal forms mean? Describe the function implemented by the ARS. A)The normal forms can be used to identify the number of equivalence classes. The function would look like a for loop iterating and returning the sum/length.

2.6 Week 9

I was a little confused with how we were suppose to specifically do the assignment. I think the hint helped to put me in the right direction but I was still unsure. We were also tasked with making milestones for our project. In the week 9 HW hint it was asked that if it was switched to = instead of - ¿ if it would change anything. From class notes it is shown "Corollary: Let $\rightarrow$ be confluent and a , b be in normal form. Then a b if and only if a = b ." So no it does not change normal forms if and only if a = b.

2.7 Week 10

We were tasked with reducing $\operatorname{fix}_F 2 this was done in class and there was also an online video that outlined how to do it.$

F is

$\lambda f. \lambda n$ if $n == 0$ then n 1 else $f(n-1) \cdot n$

$fix_F \ 2 \simeq$

$F \ fix_F \ 2 \simeq$

$\simeq (\ \lambda n.$ if $n == 0$ then 1 else $fix_F(n-1)n)2$

$\simeq (\ $ if $2 == 0$ then 1 else $fix_F(2-1)2)$

$\simeq (fix_F(1)) * 2$

$\simeq (\lambda n.$ if $n == 0$ then 1 else $fix_F(n-1)n)2$

$\simeq ($ if $1 == 0$ then 1 else $fix_F(1-1)1)2$

$\simeq (Fix_F(0))2$

$\simeq (\lambda n.$ if $n == 0$ then 1 else $fix_F(n-1)n)2$

$\simeq ($ if $0 == 0$ then 1 else $fix_F(n-1)n)2$

$= 1 * 2$

$= 2$

---

3 Project

3.1 Project Outline

I decided to change the outlook of my project and pivot to study another programming language. I originally was going to do a graphical project like plotting Mandelbrot sets but I think that I will be able to show more progress if I apply what we have learned to another language. Now describing the my new project. I originally thought I was going to do a visual project, something like plotting Mandelbrot sets but instead I am going to change to learning and explaining a new programming language Rust. My milestones will include

- Going over the history of the language. Mainly who made it, why it was made, and how it was made. (I think that I will be able to finish this section within the next 2 weeks)

- S short introduction into how to get started in the language. I would like to go over a short snippet of code that maybe I have made and how the language interprets different data types and structures. (I

think that I could finish this within 3 or 4 weeks seeing as I most likely need to be comfortable with the language to do this section)

- It's pros and cons, and it's real world application. I would like to go over the trade offs of the language and why people are starting to use this over others. (I think I can complete this with the first mile stone within the first 2/3 weeks.)

- I would also like to talk about how the language is connected to other languages and my experience with it. This is not as important to me as the other mile stones. I think it would be good to round it off and talk about how it connects in actual workflows and working with other languages. (I should be able to complete this within 2 weeks but I do not this is as important of a section and I think I should come back to it if I have time.)

If there are any suggestions on topics I should cover or another language that might be good to do the project in please let me know.

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.