

# CPSC-354 Report

Marc Domingo  
Chapman University

October 6, 2022

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Week 1 . . . . .	1
1.2	Week 2 . . . . .	1
1.3	Week 3 . . . . .	2
1.4	Week 4 . . . . .	2
1.5	Week 5 . . . . .	2
<b>2</b>	<b>Homework</b>	<b>2</b>
2.1	Week 1 . . . . .	2
2.2	Week 2 . . . . .	3
2.3	Week 3 . . . . .	5
2.4	Week 4 . . . . .	6
2.5	Week 5 . . . . .	9
2.6	Week 6 . . . . .	14
<b>3</b>	<b>Project</b>	<b>15</b>
3.1	Specification . . . . .	15
3.2	Prototype . . . . .	15
3.3	Documentation . . . . .	15
3.4	Critical Appraisal . . . . .	15
<b>4</b>	<b>Conclusions</b>	<b>15</b>

## 1 Introduction

### 1.1 Week 1

During the first week of Programming Languages, the class lectures went over the concept of *Imperative vs Functional Programming* and *Recursive Programming* in addition to a brief introduction to the programming language **Haskell**.

### 1.2 Week 2

During the second week of Programming Languages, the class lectures went more into depth regarding techniques in how to use **Recursion** to solve problems, as well as introduced the concept of creating an *Interpreter* in Haskell.

### 1.3 Week 3

During the third week of Programming Languages, the class lectures went over the concept of Context and Parse Trees in relation to programming a calculator interpreter.

### 1.4 Week 4

During the fourth week of Programming Languages, the lectures covered the concept of Lambda Calculus, specifically about its syntax and how to parse Lambda equations.

### 1.5 Week 5

During the fifth week of Programming Languages, the lectures covered the concept of performing substitution in Lambda Calculus through pen-and-paper, and through an interpreter.

## 2 Homework

### 2.1 Week 1

C++ Code for Greatest Common Denominator:

---

```
#include <iostream>
using namespace std;

int gcd(int a, int b)
{
    if (a == 0)
    {
        return b;
    }

    if (b == 0)
    {
        return a;
    }

    if (a == b)
    {
        return a;
    }

    if (a > b)
    {
        return gcd((a - b), b);
    }

    if (b > a)
    {
        return gcd(a, (b - a));
    }
}

int main() {
    // Write C++ code here
    cout << "GCD of 9 and 33 is : " << gcd(9, 33);
```

---

```

    return 0;
}

```

---

The function for calculating the Greatest Common Divisor (GCD) functions by first taking two integers as inputs, represented by **a** and **b**. In the case of inputs like gcd(9, 33), 9 and 33 are considered to be **a** and **b** in the gcd function respectively. The function first checks to see if either integer is 0, and in the case of either integer being zero, returns the other integer entered in the function as the gcd. If neither number is zero, the function checks to see if both integers are **equal to each other**. In the case that both integers are equal, the function returns the first integer as the gcd. If neither of the previous cases are met, the function then compares integer **a** and **b**. If integer **a** is larger, the function recursively calls itself, and *integer a is replaced with (a - b)*. If integer **b** is larger, the function recursively calls itself, and *integer b is replaced with (b - a)*. The function *continues making recursive calls* with modified numbers until a case where gcd is found.

## 2.2 Week 2

---

```

len [] = 0
len (x:xs) = 1 + len xs

select_evens [] = []
select_evens [a] = []
select_evens (x:y:list) = y:(select_evens list)

select_odds [] = []
select_odds [a] = [a]
select_odds (x:y:list) = x:(select_odds list)

member _ [] = False
member n (x:xs)
| x == n = True
| otherwise = member n xs

append [] list_original = list_original
append (x:list_add) list_original = x:(append list_add list_original)

revert [] = []
revert (item:xs) = append (revert xs) [item]

less_equal [] [] = True
less_equal (x:list_one) (y:list_two) = if x <= y
                                         then less_equal list_one list_two
                                         else False

```

---

In the case of **select\_evens** ["a", "b", "c", "d", "e"]:

---

```

select_evens ["a","b","c","d","e"] =
  "b":(select_evens ["c","d","e"]) =
  "b":"d":(select_evens ["e"]) =
  "b":"d":[] =
  ["b","d"]

```

---

In the case of **select\_odds** ["a", "b", "c", "d", "e"]:

---

```

select_odds ["a","b","c","d","e"] =

```

---

```
"a":(select_odds ["c","d","e"]) =
"a":"c":(select_odds ["e"]) =
"a":"c":["e"] =
["a","c","e"]
```

---

In the case of **member 2** [5,2,6]:

---

```
member 2 [5,2,6] =
    member 2 [2,6] =      5 != 2
    True           2 == 2
```

---

In the case of **member 3** [5,2,6]:

---

```
member 3 [5,2,6] =
    member 3 [2,6] =      5 != 3
    member 3 [6] =        2 != 3
    member 3 [] =         6 != 3
    False
```

---

In the case of **append** [1,2] [3,4,5]:

---

```
append [1,2] [3,4,5] =
    1:(append [2] [3,4,5]) =
    1:(2:(append [] [3,4,5])) =
    1:(2:([3:4:5])) =
    [1,2,3,4,5]
```

---

In the case of **revert** [1,2,3]:

---

```
revert [1,2,3] =
    append (revert [2,3]) [1] =
    append (append (revert [3]) [2]) [1] =
    append (append (append (revert []) [3]) [2]) [1] =
    append (append (append [] [3]) [2]) [1] =
    append (append ([3]) [2]) [1] =
    append (3:(append [] ([2]))) [1] =
    append (3:([2]) [1] =
    append [3,2] [1] =
    3:(append [2] [1]) =
    3:(2:(append [] [1])) =
    3:(2:([1])) =
    [3,2,1]
```

---

In the case of **less\_equal** [1,2,3] [2,3,4]:

---

```
less_equal [1,2,3] [2,3,4] =
    less_equal [2,3] [3,4] =      1 <= 2
    less_equal [3] [4] =        2 <= 3
    less_equal [] [] =          3 <= 4
    True
```

---

In the case of **less\_equal** [1,2,3] [2,3,2]:

---

```
less_equal [1,2,3] [2,3,2] =
    less_equal [2,3] [3,2] =      1 <= 2
    less_equal [3] [2] =        2 <= 3
```

---

**False**

**3 > 2**

---

## 2.3 Week 3

---

```
hanoi 5 0 2
  hanoi 4 0 1
    hanoi 3 0 2
      hanoi 2 0 1
        hanoi 1 0 2 = move 0 2
        move 0 1
        hanoi 1 2 1 = move 2 1
        move 0 2
      hanoi 2 1 2
        hanoi 1 1 0 = move 1 0
        move 1 2
        hanoi 1 0 2 = move 0 2
      move 0 1
    hanoi 3 2 1
      hanoi 2 2 0
        hanoi 1 2 1 = move 2 1
        move 2 0
        hanoi 1 1 0 = move 1 0
      move 2 1
    hanoi 2 0 1
      hanoi 1 0 2 = move 0 2
      move 0 1
      hanoi 1 2 1 = move 2 1
    move 0 2
  hanoi 4 1 2
    hanoi 3 1 0
      hanoi 2 1 2
        hanoi 1 1 0 = move 1 0
        move 1 2
        hanoi 1 0 2 = move 0 2
      move 1 0
    hanoi 2 2 0
      hanoi 1 2 1 = move 2 1
      move 2 0
      hanoi 1 1 0 = move 1 0
    move 1 2
  hanoi 3 0 2
    hanoi 2 0 1
      hanoi 1 0 2 = move 0 2
      move 0 1
      hanoi 1 2 1 = move 2 1
    move 0 2
  hanoi 2 1 2
    hanoi 1 1 0 = move 1 0
    move 1 2
    hanoi 1 0 2 = move 0 2
```

---

Movement solution for Towers of Hanoi with n Disks = 5:

---

0->2

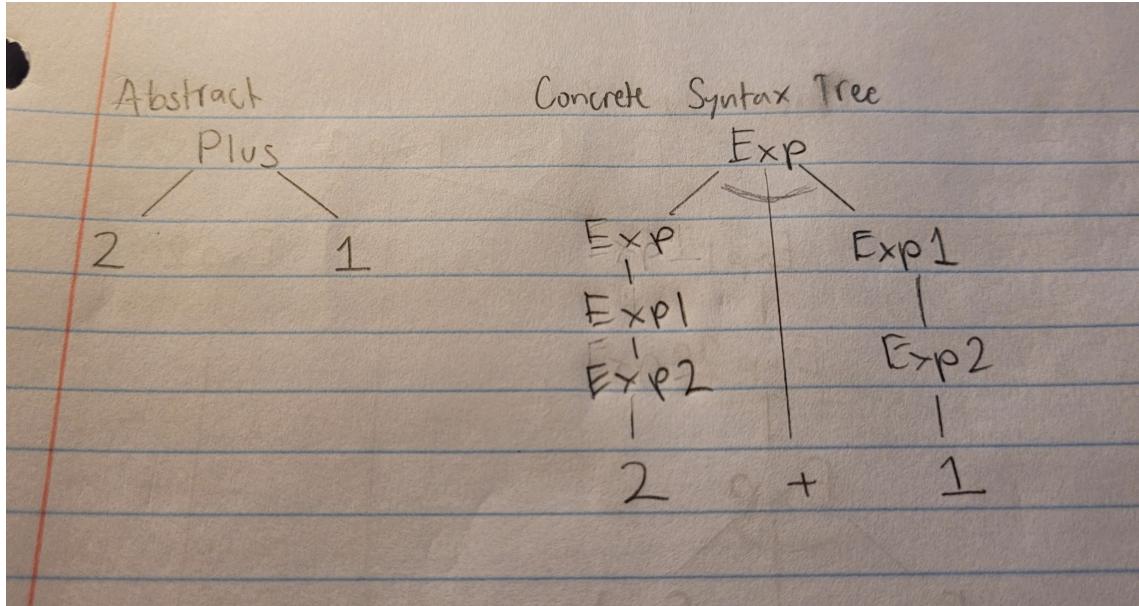
```
0->1  
2->1  
0->2  
1->0  
1->2  
0->2  
0->1  
2->1  
2->0  
1->0  
2->1  
0->2  
0->1  
2->1  
0->2  
1->0  
1->2  
0->2  
1->0  
2->1  
2->0  
1->0  
1->2  
0->2  
0->1  
2->1  
0->2  
1->0  
1->2  
0->2
```

---

Within the computation for the Towers of Hanoi with  $n$  Disks = 5, the word "Hanoi" appears a total of **31 times**. This can be expressed with the following equation:  $Hanoi = 2^N - 1$ , where  $N$  is the number of disks from the initial tower.

## 2.4 Week 4

The Abstract Syntax Tree of  $2 + 1$  is:

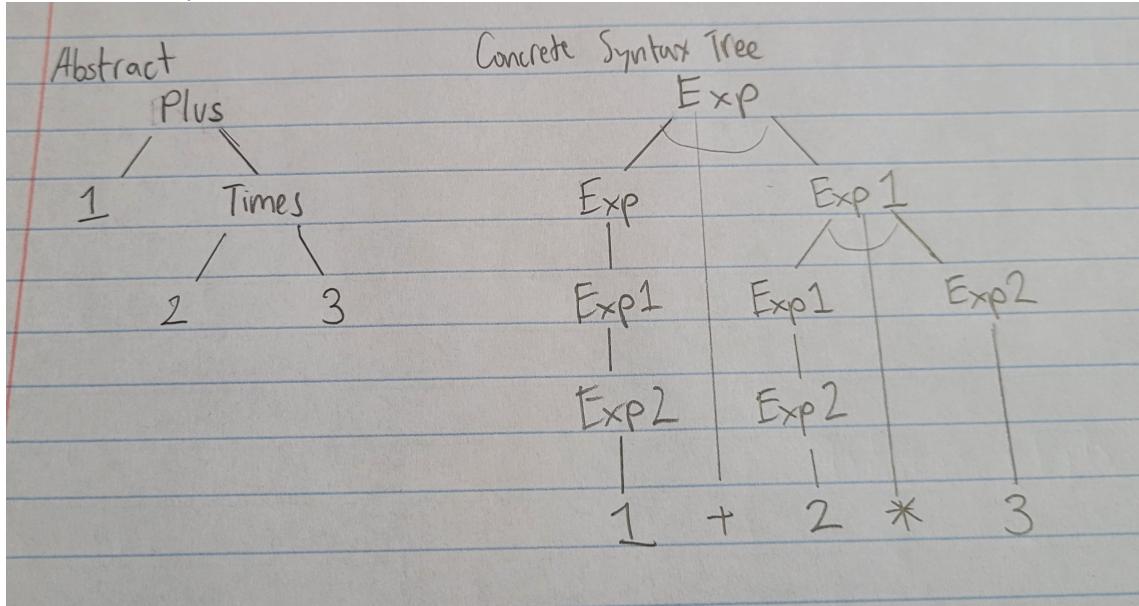



---

Plus (Num 2) (Num 1)

---

The Abstract Syntax Tree of  $1 + 2 * 3$  is:

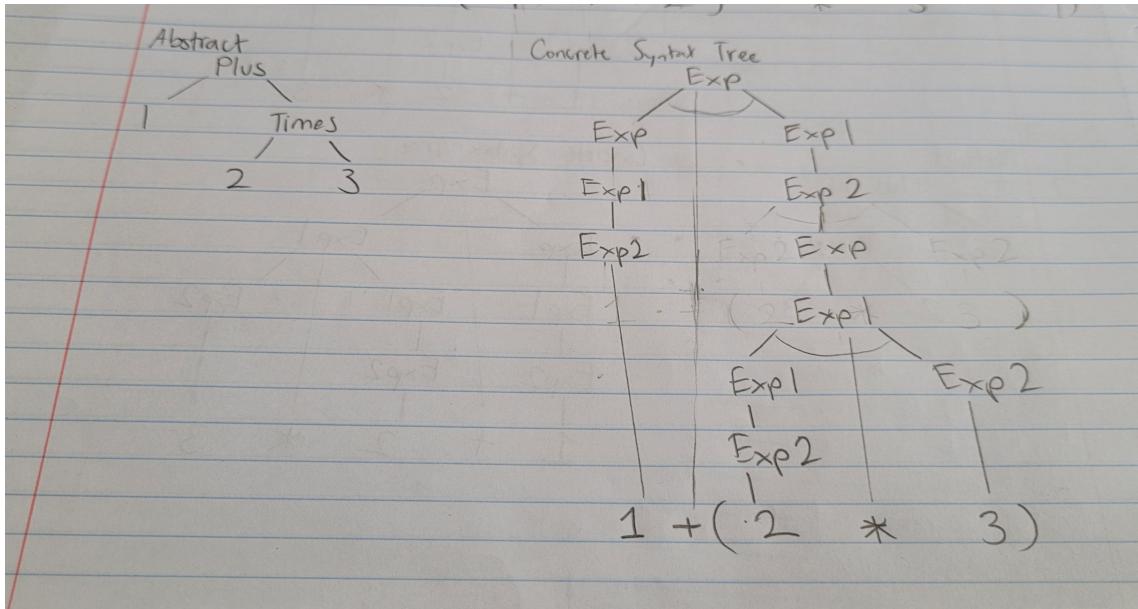



---

Plus (Num 1) (Times (Num 2) (Num 3))

---

The Abstract Syntax Tree of  $1 + (2 * 3)$  is:

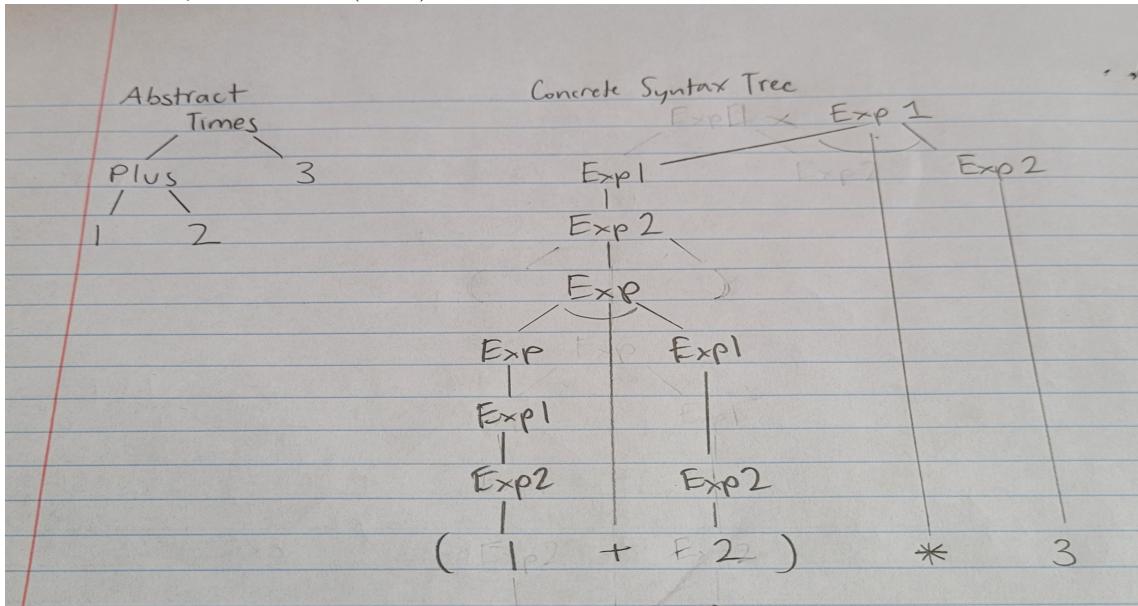



---

Plus (Num 1) (Times (Num 2) (Num 3))

---

The Abstract Syntax Tree of  $(1 + 2) * 3$  is:

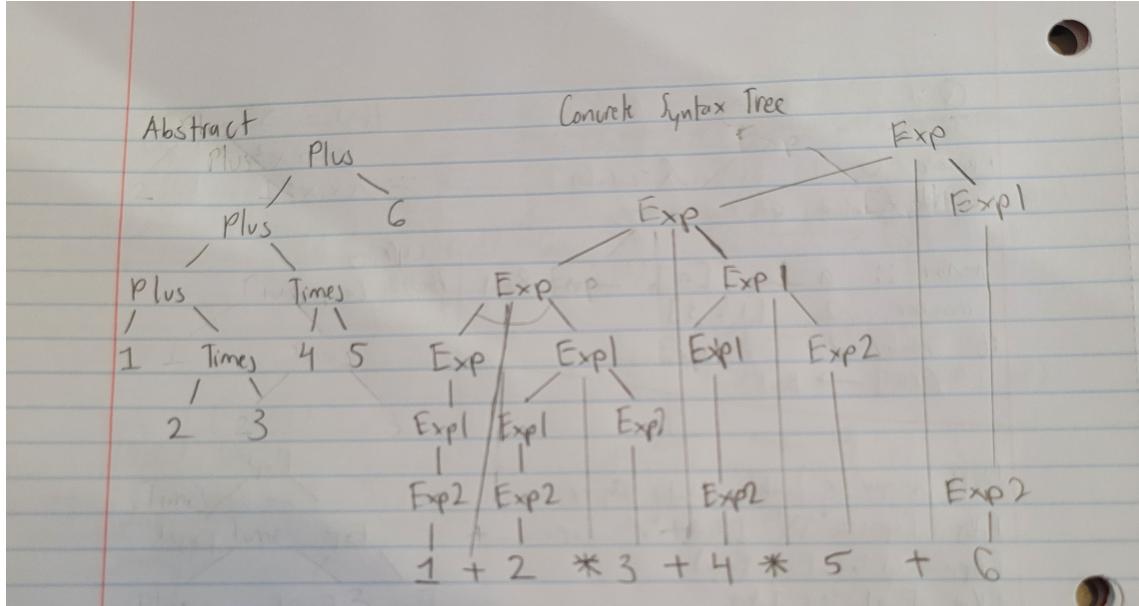



---

Times (Plus (Num 1) (Num 2)) (Num 3)

---

The Abstract Syntax Tree of  $1 + 2 * 3 + 4 * 5 + 6$  is:




---

```
Plus (Plus (Plus (Num 1) (Times (Num 2) (Num 3))) (Times (Num 4) (Num 5))) (Num 6)
```

---

The abstract syntax tree of  $1 + 2 + 3$  would be identical to the one of  $(1 + 2) + 3$  as the compiler would calculate from left to right as there are no other higher-order operations aside from addition.

## 2.5 Week 5

### Part 1:

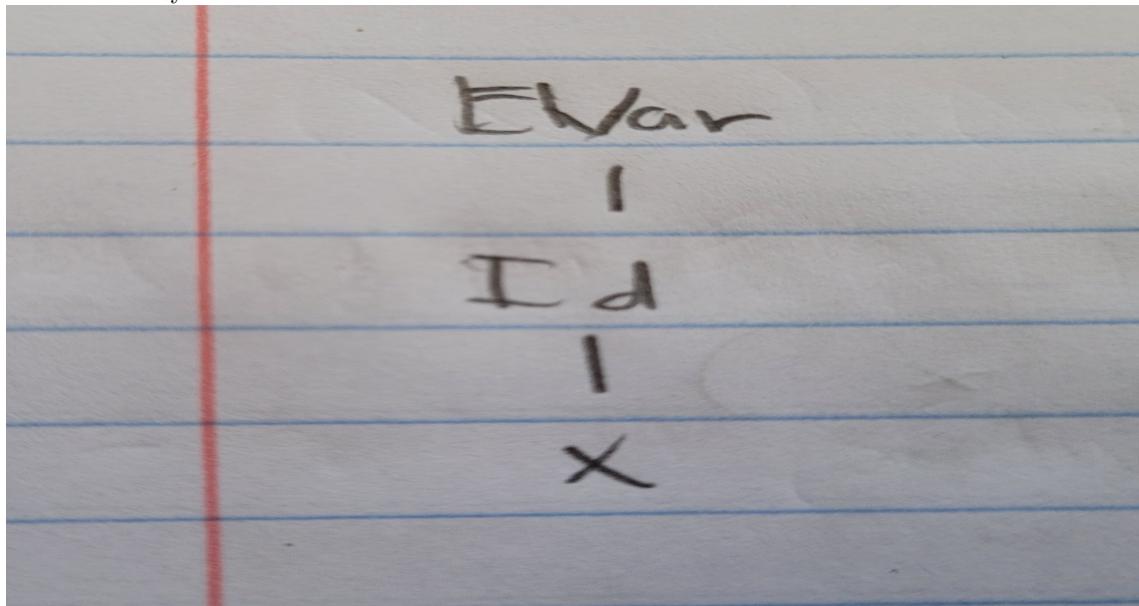
The Linearized Tree for  $x$  is:

---

x

---

Its Abstract Syntax Tree:



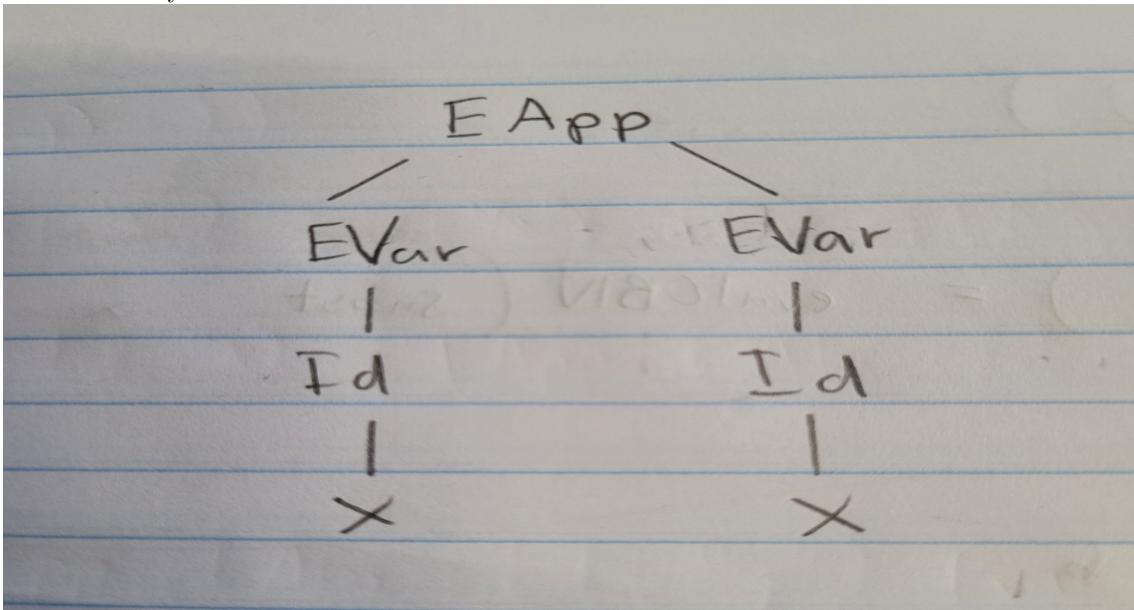
The Linearized Tree for x x is:

---

x x

---

Its Abstract Syntax Tree:



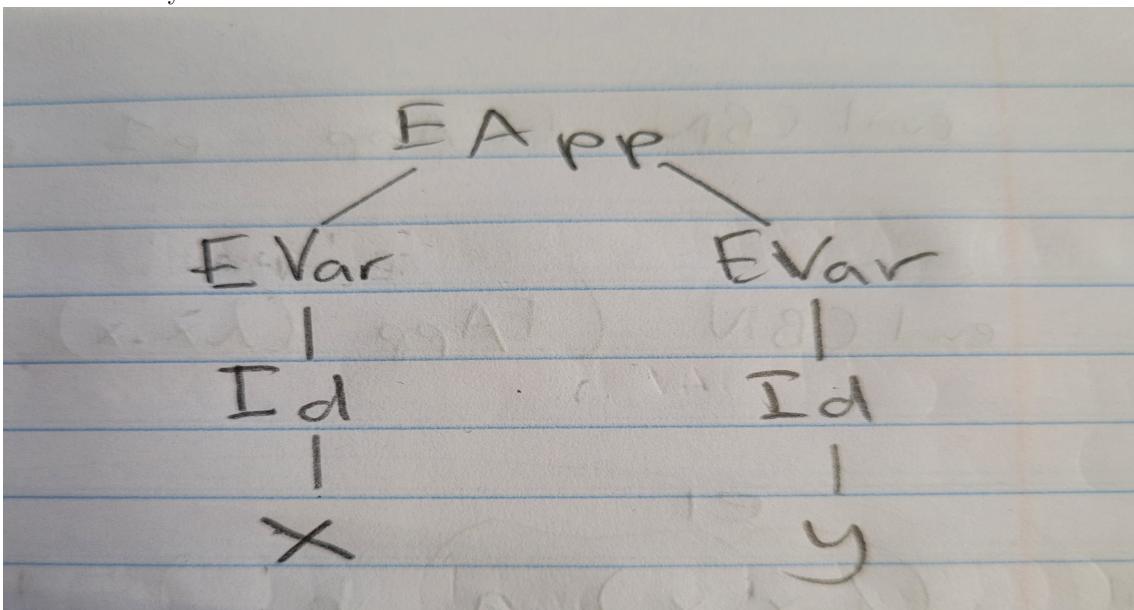
The Linearized Tree for x y is:

---

x y

---

Its Abstract Syntax Tree:



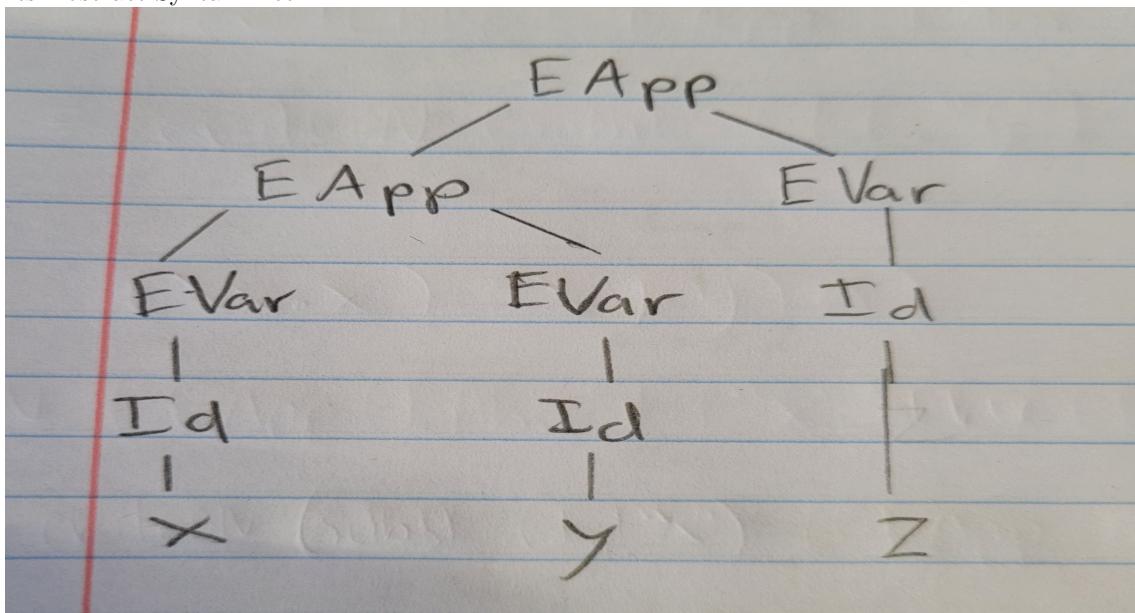
The Linearized Tree for x y z is:

---

x y z

---

Its Abstract Syntax Tree:



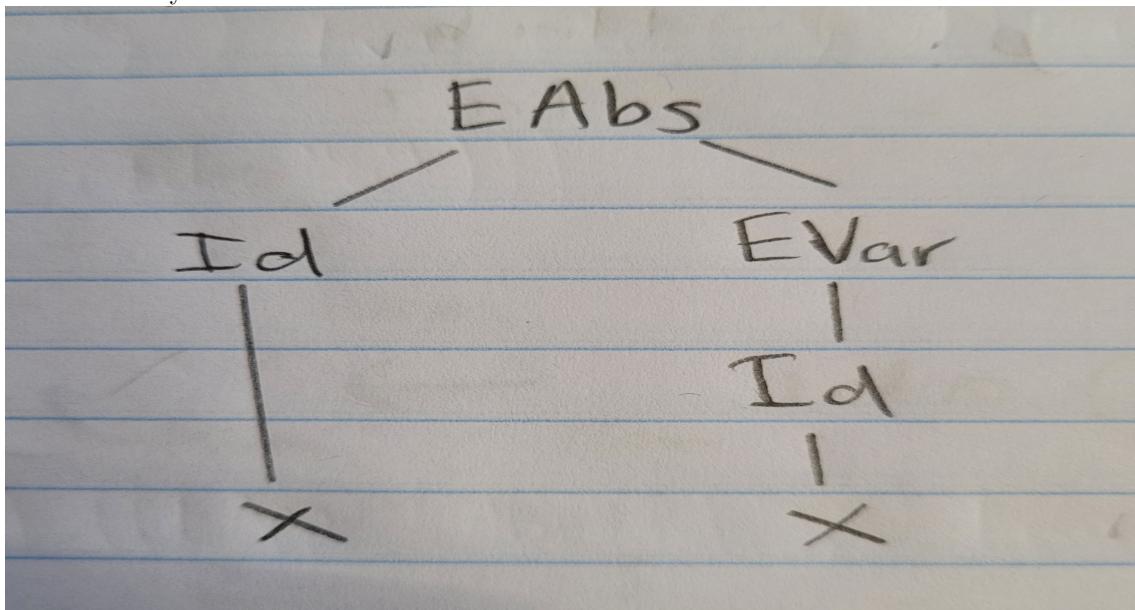
The Linearized Tree for  $\lambda x.x$  is:

---

$\lambda x . x$

---

Its Abstract Syntax Tree:



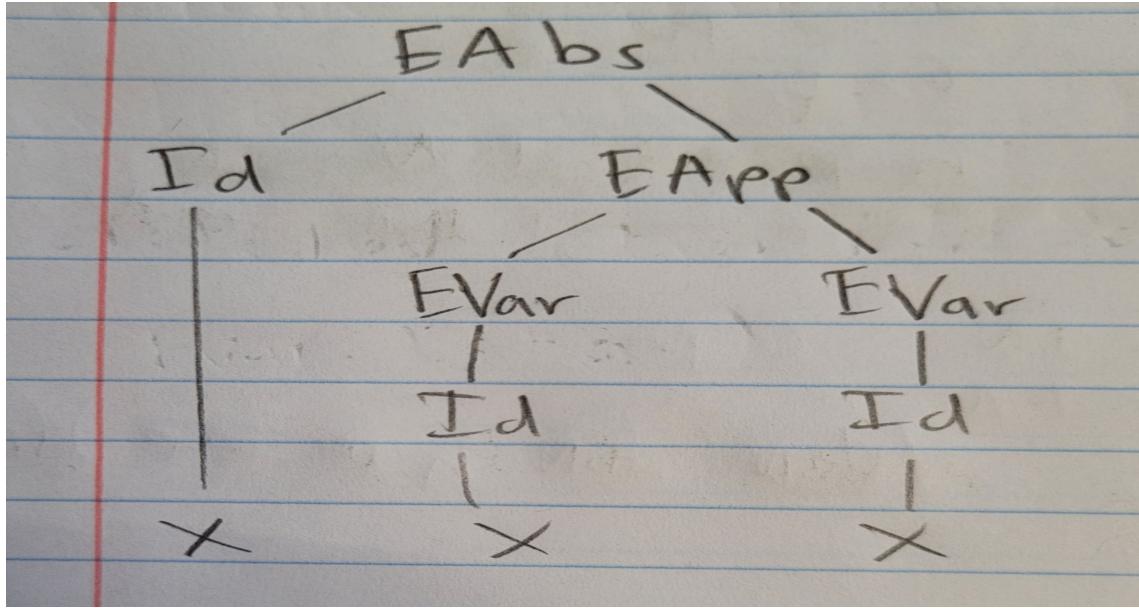
The Linearized Tree for  $\lambda x.x x$  is:

---

$\lambda x . x x$

---

Its Abstract Syntax Tree:



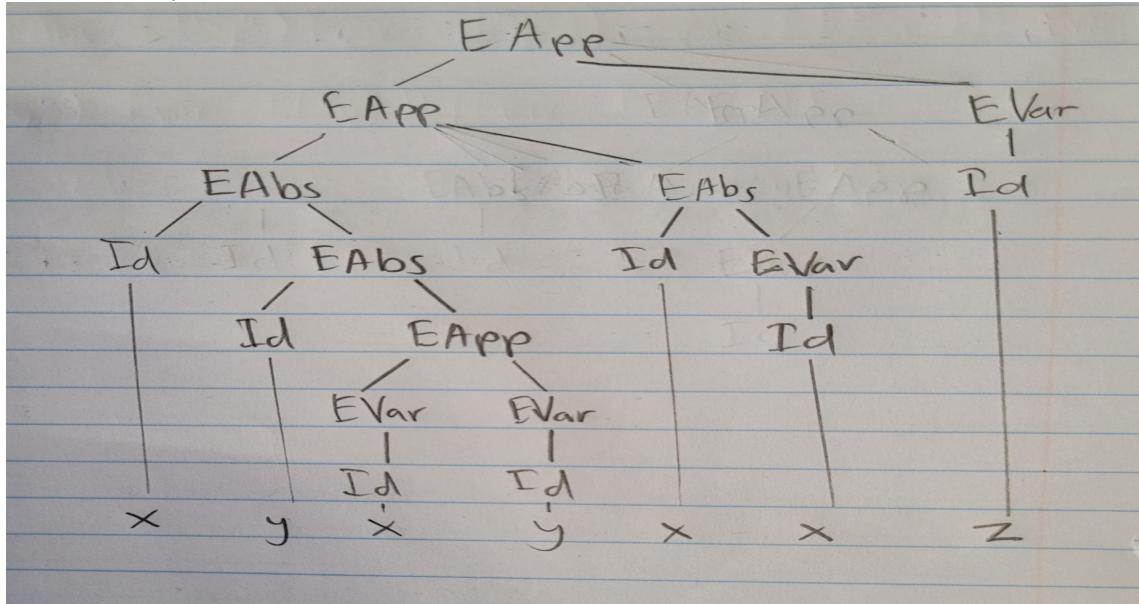
The Linearized Tree for  $(\lambda x. (\lambda y. x y)) (\lambda x. x) z$  is:

---

$\lambda x. \lambda y. x y (\lambda x. x) z$

---

Its Abstract Syntax Tree:



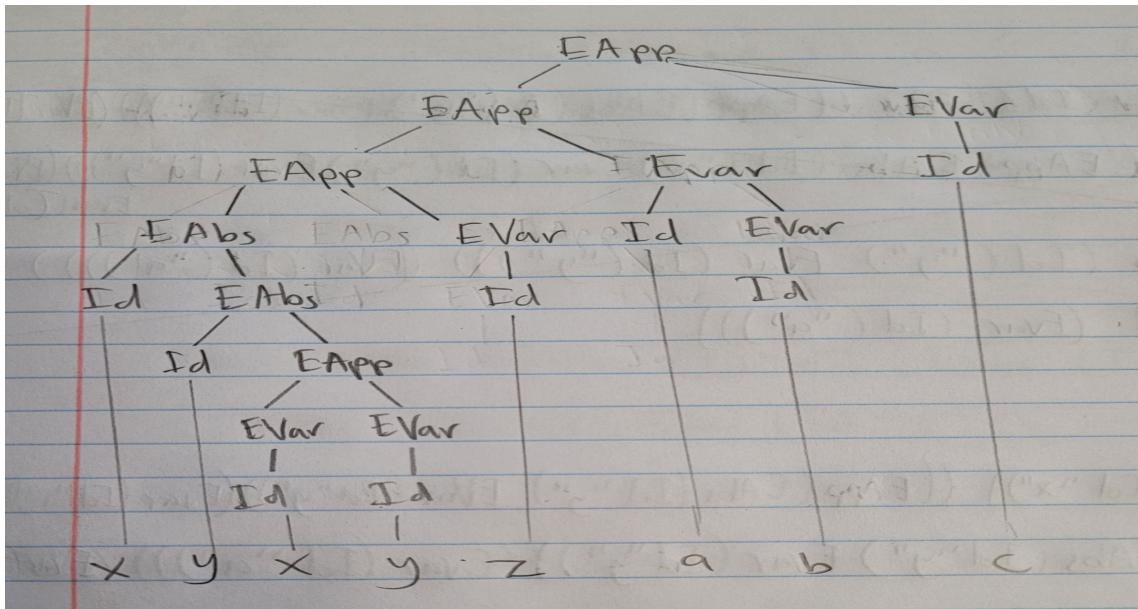
The Linearized Tree for  $(\lambda x. \lambda y. x y z) a b c$  is:

---

$\lambda x. \lambda y. x y z a b c$

---

Its Abstract Syntax Tree:



**Part 2:**

Evaluating of Equations:

---

```

(\x.x) a = a
\x.x a = a
(\x.\y.x) a b = (\y. a) b = a
(\x.\y.y) a b = (\y. y) b = b
(\x.\y.x) a b c = (\y. a) b c = (a) c = a
(\x.\y.y) a b c = (\y. y) b c = (b) c = b
(\x.\y.x) a (b c) = (\y. a) (b c) = a
(\x.\y.y) a (b c) = (\y. y) (b c) = b c
(\x.\y.x) (a b) c = (\y. a b) c = a b
(\x.\y.y) (a b) c = (\y. y) c = c
(\x.\y.x) (a b c) = (\y. a b c)
(\x.\y.y) (a b c) = (\y. y)

```

---

$(\lambda x.x)((\lambda y.y)a)$   
 $\text{evalCBN } (\text{EApp}(\text{EAb}(Id^"x") \text{Evar}(Id^"x")) ((\text{EApp}(\text{EAb}(Id^"y") \text{EVar}(Id^"y")) (\text{Evar}(Id^"a")))))$   
 $= \text{evalCBN } (\text{subst}(Id^"x") ((\text{EApp}(\text{EAb}(Id^"y") \text{Evar}(Id^"y")) (\text{Evar}(Id^"a")))) (\text{Evar}(Id^"x")))$   
 $= \text{evalCBN } (\text{EApp}(\text{EAb}(Id^"y") \text{Evar}(Id^"y")) (\text{Evar}(Id^"a")))$   
 $= \text{evalCBN } (\text{subst}(Id^"y") (\text{EVar}(Id^"a")) (\text{EVar}(Id^"y")))$   
 $= \text{evalCBN } (\text{Evar}(Id^"a"))$   
 $= \text{EVar}(Id^"a")$

## 2.6 Week 6

---

```

(\exp . \two . \three . exp two three)
(\m.\n. m n)
(\f.\x. f (f x))
(\f.\x. f (f (f x)))

=
((\m.\n. m n) (\f2.\x2. f2 (f2 x2)) (\f3.\x3. f3 (f3 (f3 x3))))
=
((\n. (\f2.\x2. f2 (f2 x2)) n) (\f3.\x3. f3 (f3 (f3 x3))))
=
((\f2.\x2. f2 (f2 x2)) (\f3.\x3. f3 (f3 (f3 x3))))
=
(\x2. (\f3.\x3. f3 (f3 (f3 x3))) ((\f3.\x3. f3 (f3 (f3 x3))) x2))
=
(\x2. (\f3.\x3. f3 (f3 (f3 x3))) (\x3. x2 (x2 (x2 x3))))
=
(\x2. (\x3. (\x3. x2 (x2 (x2 x3))) ((\x3. x2 (x2 (x2 x3))) (\x3. x2 (x2 (x2 x3))) x3)))
=
(\x2. (\x3. (\x3. x2 (x2 (x2 x3))) ((\x3. x2 (x2 (x2 x3))) ((x2 (x2 (x2 x3)))))))
=
(\x2. (\x3. (\x3. x2 (x2 (x2 x3))) ((x2 (x2 (x2 (x2 (x2 x3)))))))))))

```

---

## 3 Project

### 3.1 Specification

For my project in Programming Languages (Fall 2022), I intend on performing data analysis on a data set that is publicly available from GitHub. To accomplish this, the following would have to occur (in order):

1. Receive approval from Dr. Kurtz for this specific project idea
2. Come up with specific questions to answer about the Programming Languages data set, the responses to which can be modeled through various statistical models such as Linear/Logistic Regression Graphs, Data Cluster Graphs, Feature Reduction, LASSO Graphs, and/or Principle Component Analysis (PCA)
3. Receive approval from Dr. Kurtz about the questions and the overall documentation of the data analysis report
4. Download or import the data set from GitHub into Google Colab, this can be accomplished by replicating a specific iteration of the GitHub Programming Languages data set in an Excel sheet
5. Using the Pandas package for Python, create statistical models to respond to each of the questions created for the report
6. By the last day of October, provide a PDF of the Google Colab Notebook containing the compiled statistical models as a proof of implementation for this project. This can also be accomplished by embedding the Python code into the "Prototype" subsection of *report.tex*, with each code block being above the respective model that it generated when compiled
7. Write report for the analysis of the data set itself, explaining the process of selecting which model(s) to accept, reject, or if any model provided any surprising results about the data set in addition to providing a response to each of the questions created from **step 4**. A rough draft of this report should be submitted and present within the "Documentation" subsection of *report.tex*

### 3.2 Prototype

### 3.3 Documentation

### 3.4 Critical Appraisal

## 4 Conclusions

...