

CPSC-354 Report

Stephanie Munday
Chapman University

October 9, 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	6
2.5	Week 5	7
2.6	Week 6	11
3	Project	11
3.1	Specification	11
3.2	Prototype	11
3.3	Documentation	11
3.4	Critical Appraisal	11
4	Conclusions	12

1 Introduction

This is the report for CPSC 354 Programming Languages. It will contain homework for each week, as well as project work and analysis.

2 Homework

This section will contain your solutions to homework.

2.1 Week 1

HW 1 - Greatest Common Divisor

```
def gcd(n, m):
    while n != m:
        if n > m:
            n = n-m
        else:
            m = m-n
    return n
```

The code above implements Euclid's algorithm to find the greatest common divisor in python. Below is an explanation given sample input `gcd(9,33)`.

While `n != m`, the code will compare whether or not `n` is greater than `m`. If `n > m`, `n` will become `n - m`. Otherwise if `n < m`, `m` will become `m - n`. When `n == m`, the greatest common divisor has been found.

Keeping this logic in mind, let `n = 9`, `m = 33`.

```
gcd(9,33) =
gcd(9,24) =
gcd(9,15) =
gcd(9,6) =
gcd(3,6) =
gcd(3,3) =
3
```

Since `n == m` and the value of both is 3, the greatest common divisor is 3 for this example.

2.2 Week 2

HW 2 - Recursion in Functional Programming

```
select_evens :: [a] -> [a]
select_evens [] = []
select_evens (x:(y:xs)) = y:select_evens(xs)

select_odds :: [a] -> [a]
select_odds [] = []
select_odds (x:(y:xs)) = x:select_odds(xs)

member :: (Eq a) => a -> [a] -> Bool
member a [] = False
member a (x:xs)
    | a == x = True
    | otherwise = a `member` xs

append :: (Ord a) => [a] -> [a] -> [a]
append [] [] = []
append [] ys = ys
append (x:xs) (ys) = x:append(xs) (ys)

revert :: [a] -> [a]
revert [] = []
revert (x:xs) = append (revert xs) [x]

less_equal :: (Ord a) => [a] -> [a] -> Bool
less_equal [] [] = True
```

```
less_equal (x:xs) (y:ys)
  | x > y   = False
  | otherwise = xs 'less_equal' ys
```

The code above implements `select_evens`, `select_odds`, `member`, `append`, `revert`, `less_equal` as recursive functions in Haskell. Below are explanations showing computations for given inputs.

Select Evens example:

Select Evens ["a","b","c","d"]

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

Select Odds example:

Select Odds ["a","b","c","d"]

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

Member example:

Member 2 [5,2,6]

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

Append example:

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

Revert example:

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 : (2) [1] =
```

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

Less Equal example:

Less Equal [1,2,3] [2,3,4]

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

2.3 Week 3

HW 3 - Towers of Hanoi

```
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

```

In order to solve the puzzle, the moves are as follows:

```

move 0 2
move 0 1
move 2 1
move 0 2
move 1 0
move 1 2
move 0 2
move 0 1
move 2 1
move 2 0
move 1 0
move 2 1
move 0 2
move 0 1
move 2 1
move 0 2
move 1 0
move 1 2
move 0 2
move 1 0
move 2 1
move 2 0
move 1 0
move 1 2
move 0 2
move 0 1
move 2 1
move 0 2
move 1 0
move 1 2
move 0 2

```

The word "hanoi" appears in the computation 31 times.

This computation can be expressed as a formula that works for moving any number of disks n as:

```

hanoi(n+1) x y = hanoi n x(other x y)
move x y
hanoi n(other x y)y

```

```

hanoi 1 x y = move x y

```

```

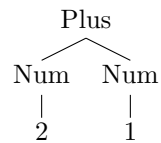
hanoi (n+1) x y =
  hanoi n x (other x y)
  move x y
  hanoi n (other x y) y

```

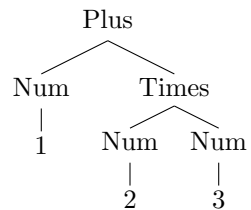
2.4 Week 4

HW 4 - Parsing and Context-Free Grammars

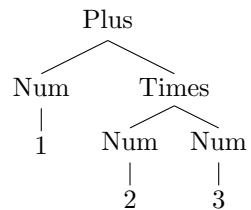
Abstract Syntax Tree: $2 + 1$
 Plus (Num 2) (Num 1)



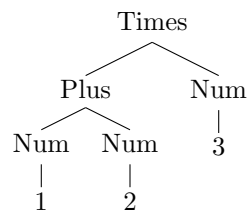
Abstract Syntax Tree: $1 + 2 * 3$
 Plus (Num 1) (Times (Num 2) (Num 3))



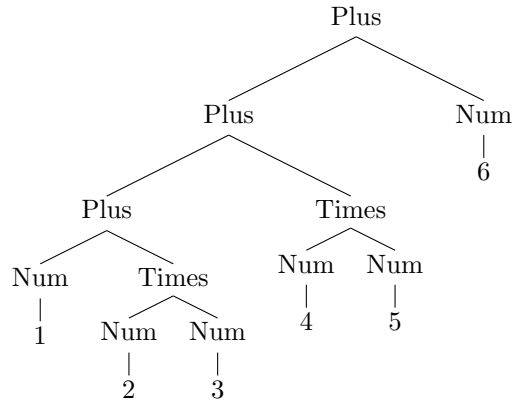
Abstract Syntax Tree: $1 + (2 * 3)$
 Plus (Num 1) (Times (Num 2) (Num 3))



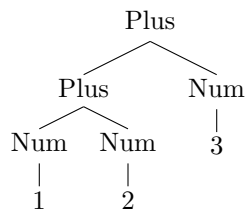
Abstract Syntax Tree: $(1 + 2) * 3$
 Times (Plus (Num 1) (Num 2)) (Num 3)



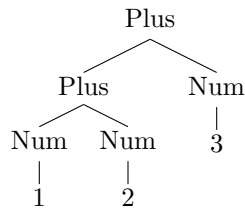
Abstract Syntax Tree: $1 + 2 * 3 + 4 * 5 + 6$
 Plus (Plus (Plus (Num 1) (Times (Num 2) (Num 3))) (Times (Num 4) (Num 5))) (Num 6)



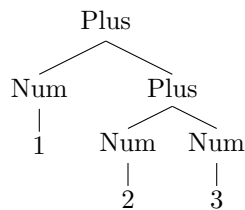
Abstract Syntax Tree: $1 + 2 + 3$
`Plus (Plus (Num 1) (Num 2)) (Num 3)`



Abstract Syntax Tree: $(1 + 2) + 3$
`Plus (Plus (Num 1) (Num 2)) (Num 3)`



Abstract Syntax Tree: $1 + (2 + 3)$
`Plus (Num 1) (Plus (Num 2) (Num 3))`

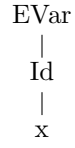


The abstract syntax tree of $1+2+3$ is identical to the one of $(1+2)+3$, but **not** the one of $1+(2+3)$.

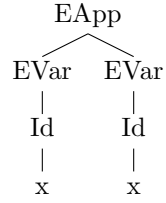
2.5 Week 5

HW 5 - Syntax + Semantics of Lambda Calculus Syntax

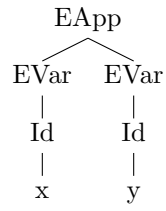
`x = EVar (Id "x")`



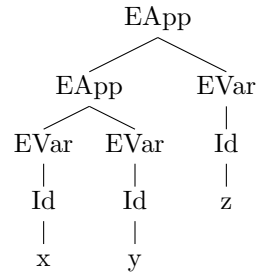
`x x = EApp (EVar (Id "x") EVar (Id "x"))`



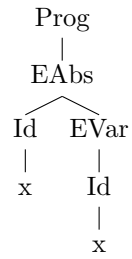
`x y = EApp (EVar (Id "x") EVar (Id "y"))`



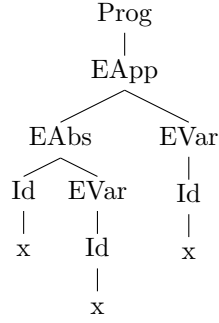
`x y z = EApp (EVar (Id "x") EVar (Id "y")) EVar (Id "z")`



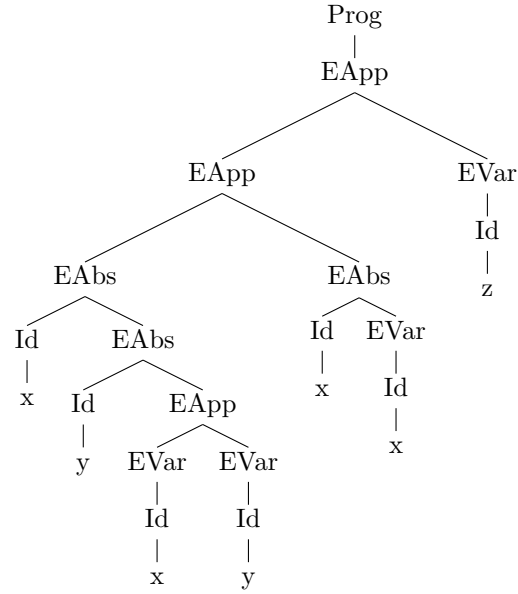
`\ x.x = Prog (EAbs(Id "x" EVar(Id "x")))`



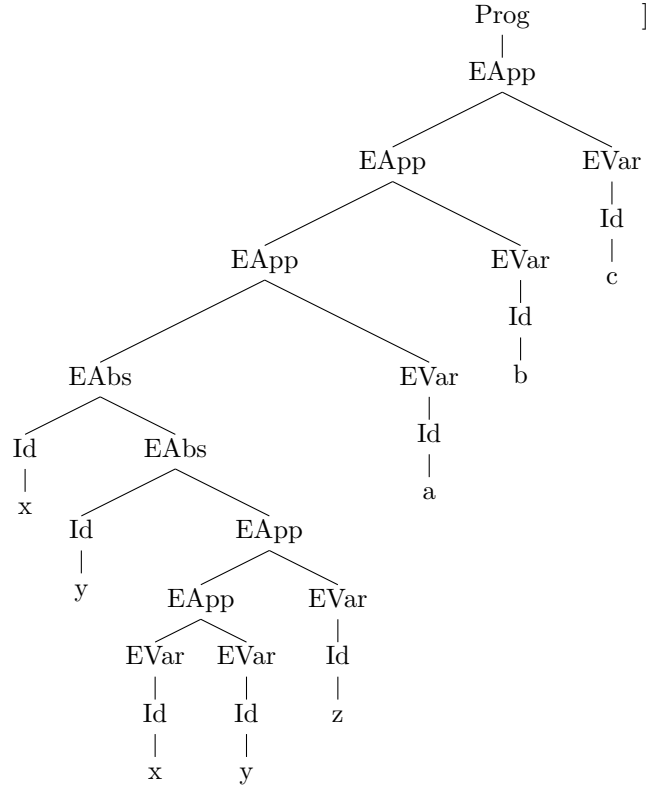
`(\x.x) x = Prog(EApp(EAbs(Id "x" EVar(Id "x")) EVar(Id "x")))`



$(\lambda x . (\lambda y . x y)) (\lambda x.x) z = \text{Prog}(\text{EApp}(\text{EApp}(\text{EAbs}(\text{Id } \text{"x"}), \text{EAbs}(\text{Id } \text{"y"}), \text{EApp}(\text{EVar}(\text{Id } \text{"x"}), \text{EVar}(\text{Id } \text{"y"})))), \text{EAbs}(\text{Id } \text{"x"}, \text{EVar}(\text{Id } \text{"x"}))), \text{EVar}(\text{Id } \text{"z"}))$



$(\lambda x . \lambda y . x y z) a b c = \text{Prog}(\text{EApp}(\text{EApp}(\text{EApp}(\text{EAbs}(\text{Id } \text{"x"}), \text{EAbs}(\text{Id } \text{"y"}), \text{EApp}(\text{EApp}(\text{EVar}(\text{Id } \text{"x"}), \text{EVar}(\text{Id } \text{"y"}))), \text{EVar}(\text{Id } \text{"z"}))), \text{EVar}(\text{Id } \text{"a"}), \text{EVar}(\text{Id } \text{"b"}), \text{EVar}(\text{Id } \text{"c"}))$



Semantics

- Evaluate using pen-**and**-paper the following expressions:

$(\lambda x.x) a = a$

$\lambda x.x a = \lambda x.x a$

$(\lambda x.\lambda y.x) a b = (\lambda y.a) b = a$

$(\lambda x.\lambda y.y) a b = (\lambda y.y) b = b$

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

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

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

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

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

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

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

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

```
- Evaluate (\x.x)((\y.y)a) by executing the function evalCBN

evalCBN(EApp (EAbs (Id "x") (EVar (Id "x")))) (EApp (EAbs (Id "y") (EVar (Id "y")))) (EVar (Id
  "a")))) =
evalCBN (EApp (EAbs (Id "x") (EVar (Id "x")))) subst (Id "y") (EVar (Id "a")) (EVar (Id "y"))) =
evalCBN (EApp (EAbs (Id "x") (EVar (Id "x")))) EVar (Id "a")) =
evalCBN (subst (Id "x") (EVar (Id "a")) (EVar (Id "x"))) =
evalCBN (EVar (Id "a")) =
EVar (Id "a")
```

2.6 Week 6

Evaluate

```
(\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) (\f.\x. f (f x)) (\f2.\x2. f2 (f2 (f2 x2))))
=
((.\n. (\f.\x. f (f x)) n) (\f2.\x2. f2 (f2 (f2 x2))))
=
((\f.\x. f (f x)) (\f2.\x2. f2 (f2 (f2 x2))))
=
((\x. (\f2.\x2. f2 (f2 (f2 x2))) (\f3.\x3. f3 (f3 (f3 x3))) x)))
=
((\x. (\f2.\x2. f2 (f2 (f2 x2))) (\x3. x (x (x x3)))))
=
(\x. (\x2. (\x3. x (x (x x3))) (\x4. x5 (x5 (x5 x4))) (\x6. x7 (x7 (x7 x6))) x2)))
=
(\x. (\x2. (\x3. x (x (x x3))) (\x4. x5 (x5 (x5 x4))) (x7 (x7 (x7 x2))))))
=
(\x. (\x2. (x (x (x (x5 (x5 (x5 (x7 (x7 (x7 x2)))))))))))
=
\x. (\x2. (x (x (x (x5 (x5 (x5 (x7 (x7 (x7 x2))))))))))
```

3 Project

This section details the project.

3.1 Specification

For this project, I plan to learn a combination of HTML, javascript, and css to build a portfolio website.

3.2 Prototype

3.3 Documentation

3.4 Critical Appraisal

...

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.

[P] [Punctuation](#), StackExchange, 2022.

[S] [Spacing](#), StackExchange, 2022.

[T] [Trees](#), Massachusetts Institute of Technology, 2022.