

CPSC-354 Report

Tyler Lewis
Chapman University

October 3, 2022

Abstract

Updated throughout Fall 2022 for 354 Programming Languages at Chapman Univ.

Contents

1	Introduction	1
2	Homework	1
2.1	Week 1	1
2.2	Week 2	2
2.3	Week 3	3
2.4	Week 4	4
2.5	Week 5 (line 350)	6
3	Project	9
3.1	Specification	9
3.2	Prototype	9
3.3	Documentation	9
3.4	Critical Appraisal	9
4	Conclusions	9

1 Introduction

Tylers introduction. Yeah, this will get some work before final submission.

2 Homework

2.1 Week 1

Euclid's Algorithm

Input: Two whole numbers (integers) called a and b , both greater than 0.

- (1) if $a < b$ then replace a by $(a - b)$.
- (2) if $b > a$ then replace b by $(b - a)$.
- (3) Repeat from (1) if $a \neq b$

Output: a.

Code (Golang)

```
package main
import ( "fmt"; "strconv"; "os" )
// Calculate GCD of a & b using Euclid's algorithm
func Euclid-GCD( a int, b int ) int {
    if a > b { return Euclid_GCD( a-b, b ) } // recursive GCD function
    if a < b { return Euclid_GCD( a, b-a ) } // Subtract lesser from greater
    return a // a == b End recursive function
} func main() {
    // Args(str) int conversion
    a, err1 := strconv.Atoi(os.Args[1]); b, err2 := strconv.Atoi(os.Args[2])
    // If no errors:
    if err1 == nil && err2 == nil {
        gcd := Euclid_GCD( a, b ) // Evaluate GCD of args(int) => a, b
        fmt.Println(gcd) // Print divisor to console
        return // End script
    } fmt.Println("Error", err1, err2) // Errors happened
}
```

Explanation

Following the steps of Euclid's algorithm detailed in section **Euclid's Algorithm**, the GCD between any two numbers is determined. The Golang function, **Euclid-GCD**, detailed step-by-step in section **Code (Golang)**, determines the GCD by recursively subtracting one non-zero integer by the other.

How to run:

1-3 need only be done once:

- (1) Install Golang
- (2) Init Golang project: `go mod init`
- (3) Compile: `go build gcd.go`
- (4) Run: `./gcd.go [int arg1] [int arg2]`

2.2 Week 2

Task 1

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

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

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

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

Task 2

```
append [2,5,4,3] 5
-> [2]:[5]:[4]:[3]: 5
-> [2,5,4,3,5]
```

2.3 Week 3

Completed 'fill in the dot' execution:

```
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
    |
  |
|

```

The word 'hanoi' appears 31 times for a tower of height 5. Hanoi will execute $\{2^n - 1\}$ times

Javascript-ish formula to solve Tower of Hanoi with n discs:

```

func hanoi( n, x, y ) {
  switch( n ) {
    case 1:
      move( x, y );
      break;
    default:
      hanoi ( n-1, x, other( x, y ) );
      move( x, y );
      hanoi ( n-1, other( x, y ), y );
      break;
  }
}

func move( x, y ) {
  // move top disk of position x to position y
}

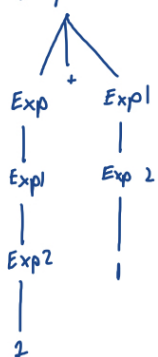
func other( x, y ) {
  return (2 * ( x + y )) % 3;
}

```

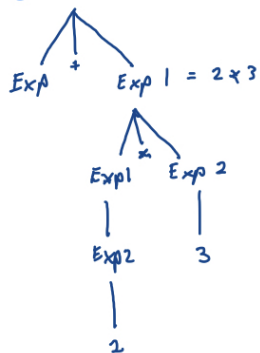
2.4 Week 4

derivation trees

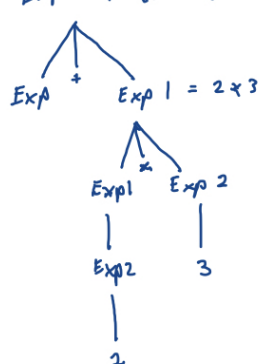
① $Exp = 2 + 1$



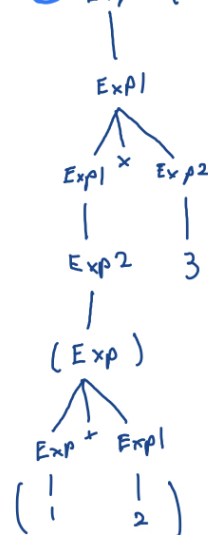
② $Exp = 1 + 2 * 3$



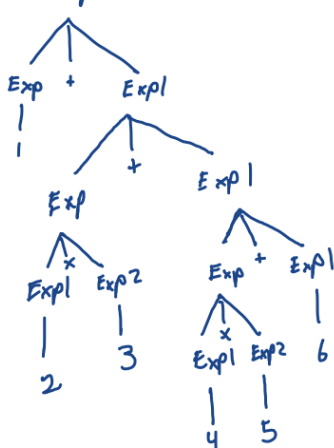
③ $Exp = 1 + (2 * 3)$



④ $Exp = (1 + 2) * 3$



⑤ $Exp = 1 + 2 * 3 + 4 * 5 + 6$



$= 1 + (2 * 3) + (4 * 5) + 6$

"More exercises"

Why do the following strings not have parse trees (given the context-free grammar above)?

2-1: No rule for subtraction

1.0+2: Only rules for integers

6/3: No specification for division

8 mod 6: No specification for modulus

Can you change the grammar, so that the strings in the previous exercise become parsable?

yes you can, I would assume for modulus as well

write out the abstract syntax trees for the following strings:

2+1: Plus (Num 2) (Num 1)

1+2*3: Plus (Num 1) (Times (Num 2) (Num 3))

1+(2*3): Plus (Num 1) (Times (Num 2) (Num 3))

(1+2)*3: Times (Plus (Num 1) (Num 2)) (Num 3)

Is the abstract syntax tree of $1+2+3$ identical to the one of $(1+2)+3$ or the one of $1+(2+3)$?

No particular right answer.

2.5 Week 5 (line 350)

Use the parser to generate linearized abstract syntax trees for the following expressions:

x

`Prog (EVar (Id "x"))`

$x\ x$

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

$x\ y$

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

$x\ y\ z$

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

$\lambda x.x$

`Prog (EAbs (Id "x") (EVar (Id "x")))`

$\lambda x.x\ x$

`Prog (EAbs (Id "x") (EApp (EVar (Id "x")) (EVar (Id "x"))))`

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

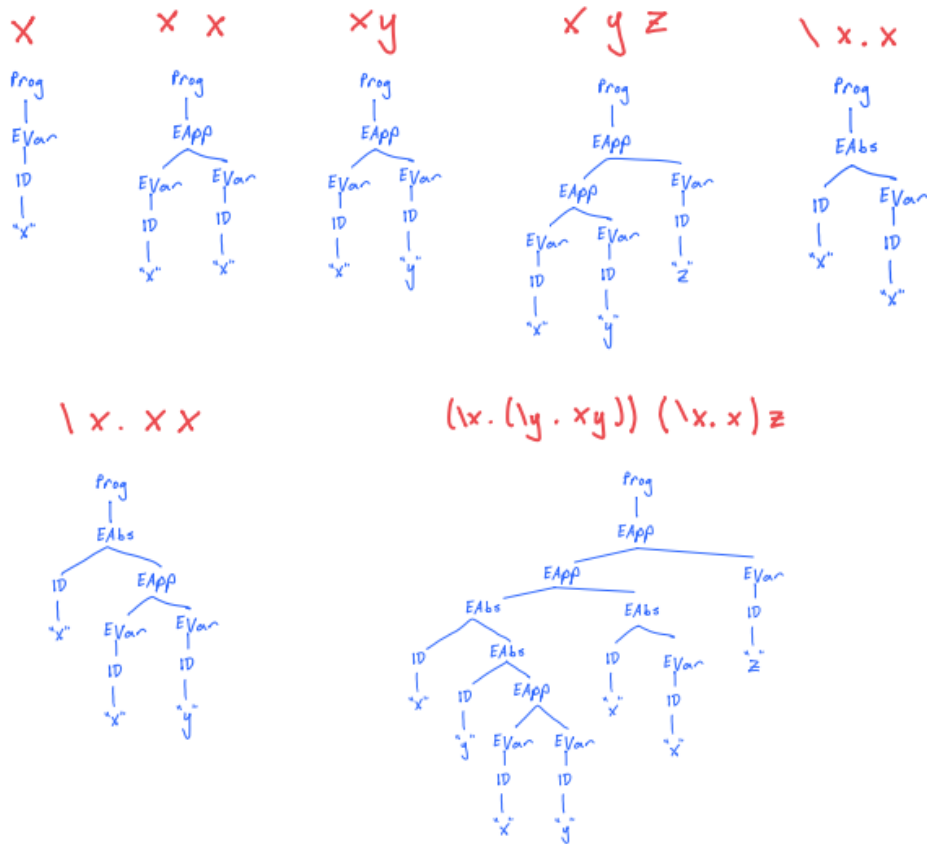
`Prog (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EApp (EVar (Id "x")) (EVar (Id "y")))))) (EAbs (Id "x") (EVar (Id "x")))) (EVar (Id "z")))`

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

`Prog (EApp (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EApp (EApp (EVar (Id "x")) (EVar (Id "y")))) (EVar (Id "z"))))) (EVar (Id "a"))) (EVar (Id "b"))) (EVar (Id "c")))`

Write out the abstract syntax trees in 2-dimensional notation using pen and paper.

2D abstract syntax trees



Evaluate using pen-and-paper [2] the following expressions:

Lambda Calculus Semantics

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

$$\lambda x. x a \longrightarrow \lambda y. y a$$

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

$$\quad \quad \quad \hookrightarrow a$$

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

$$\quad \quad \quad \hookrightarrow b$$

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

$$\quad \quad \quad \hookrightarrow a$$

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

$$\quad \quad \quad \hookrightarrow b$$

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

$$\quad \quad \quad \hookrightarrow a$$

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

$$\quad \quad \quad \hookrightarrow (b c)$$

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

$$\quad \quad \quad \hookrightarrow (a b)$$

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

$$\quad \quad \quad \hookrightarrow c$$

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

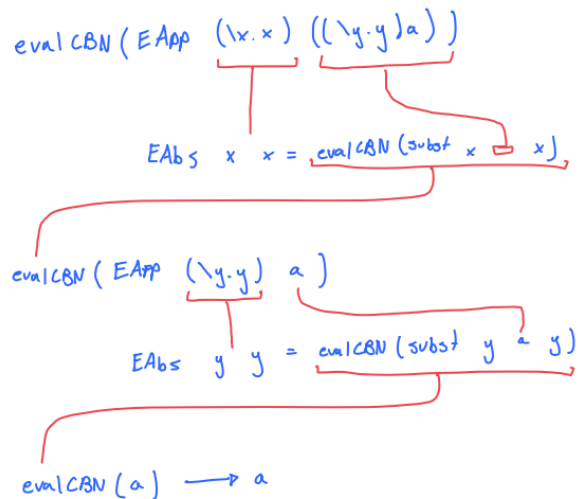
$$\quad \quad \quad \hookrightarrow (a b c)$$

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

$$\quad \quad \quad \hookrightarrow$$

Evaluate $(.x)((.y)a)$ by executing the function `evalCBN` defined on line 26-28 in `Interpreter.hs` pen-and-paper. The function `subst` is doing capture avoiding substitution and you can reduce `subst` in one step in your pen and paper computation

EvalCBN
 $(\lambda x. x) ((\lambda y. y) a)$



3 Project

Introductory remarks ...

The following structure should be suitable for most practical projects.

3.1 Specification

3.2 Prototype

3.3 Documentation

3.4 Critical Appraisal

...

4 Conclusions

Thanks, goodbye.

References

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