CPSC-354 Report

Tyler Lewis
Chapman University

November 28, 2022

Abstract

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

Contents

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)

```go
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
}
```

Explaination

Following the steps of Euclids 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:

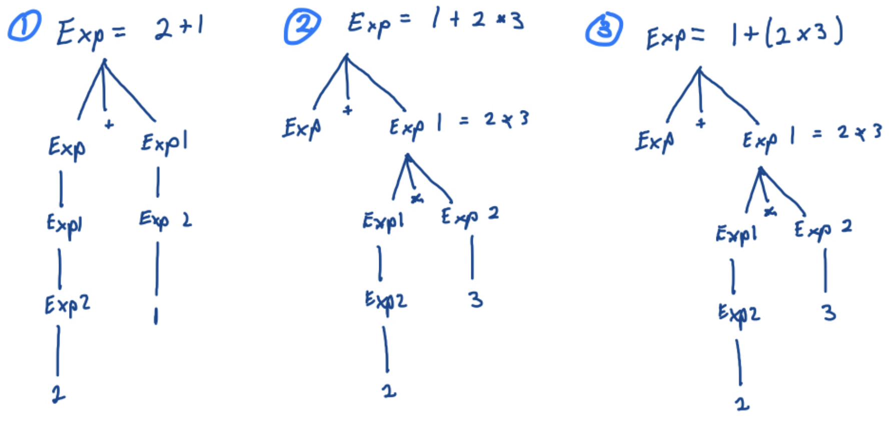
```
hanoi 5 0 2
   hanoi 4 0 1
      hanoi 3 0 2
         hanoi 2 0 1
            hanoi 1 0 2 = move 0 2
            move  0 1
            hanoi 1 2 1 = move 2 1
         move 0 2
         hanoi 2 1 2
            hanoi 1 1 0 = move 1 0
            move  1 2
            hanoi 1 0 2 = move 0 2
      move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

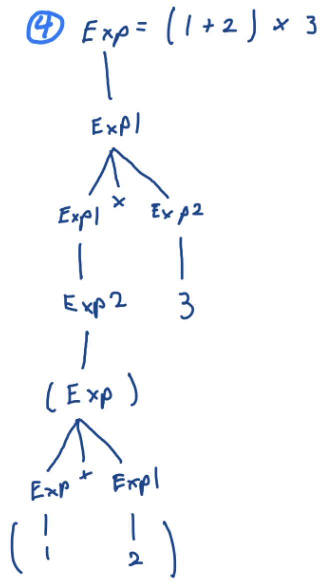
```
The word 'hanoi' appears 31 times for a tower of height 5. Hanoi will execute {2^n - 1}  times

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

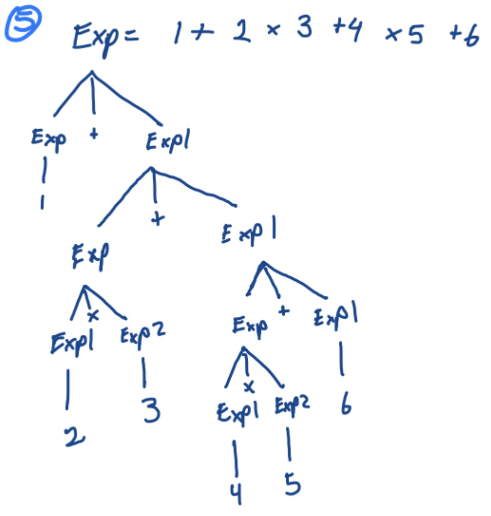
```
func hanoi( n, x, y ) {
   switch( n ) {
      case 1:
         move( x, y );
         break;
      default:
         hanoi ( n-1, x, other( x, y ) );
         move( x, y );
         hanoi ( n-1, other( x, y ), y );
         break;
   }
}

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







$= 1 + (2\times3) + (4\times5) + 6$

"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 300)

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")))
```

\x.x

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

\x.x x

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

(\x . (\y . x y)) (\x.x) z

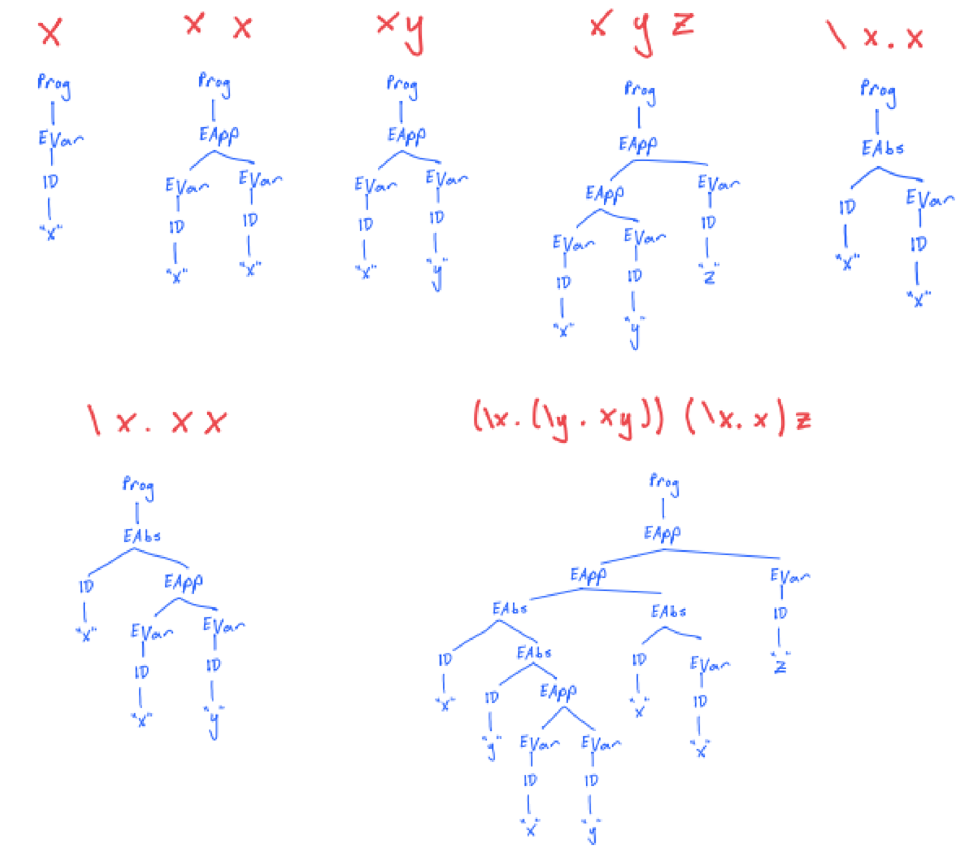
```
Prog (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EApp (EVar (Id "x")) (EVar (Id "y"))))) (EAbs (Id "x")
(EVar (Id "x")))) (EVar (Id "z")))
```

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

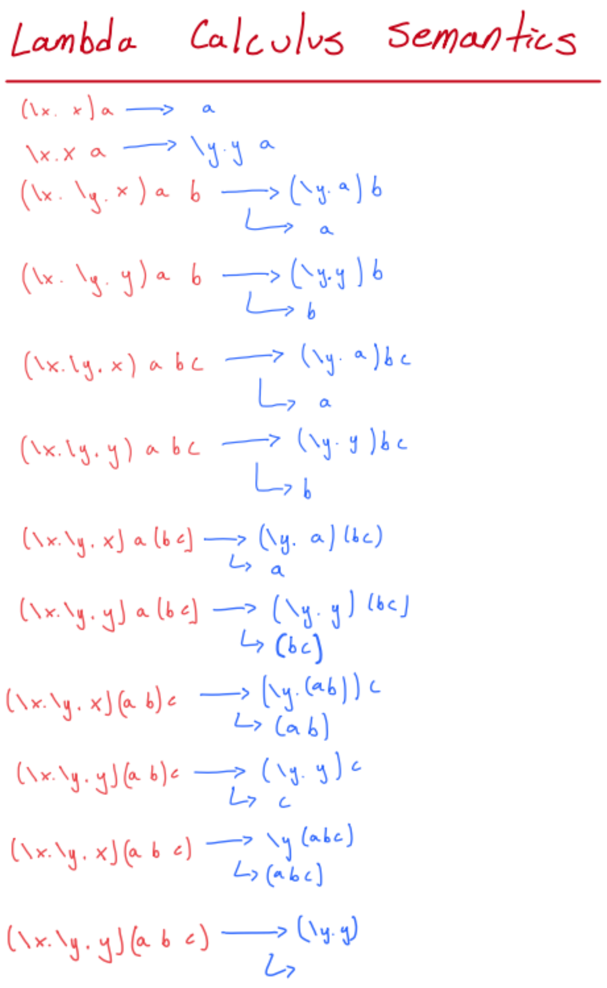
```
Prog (EApp (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EApp (EApp (EVar (Id "x"))
(EVar (Id "y"))) (EVar (Id "z"))))) (EVar (Id "a"))) (EVar (Id "b"))) (EVar (Id "c")))
```

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

```
(\x. x) a ⟶ a
\x.x a ⟶ \y.y 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
(\x.\y. y) a b c ⟶ (\y. y) b c
                    ↳ b
(\x.\y. x) a (b c) ⟶ (\y. a) (bc)
                  ↳ a
(\x.\y. y) a (b c) ⟶ (\y. y) (bc)
                    ↳ (bc)
(\x.\y. x) (a b) c ⟶ (\y. (ab)) c
                   ↳ (a b)
(\x.\y. y) (a b) c ⟶ (\y. y) c
                   ↳ c
(\x.\y. x) (a b c) ⟶ \y (abc)
                    ↳ (abc)
(\x.\y. y) (a b c) ⟶ (\y. y)
                      ↳
```

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

2.6 Week 6 (line 350)

```
Reduce the following lambda calculus expression:

    (\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)) (\f.\x. f (f (f x))) )  -- Substitution

    ( (\m.\n. m n) (\f.\x. f (f x)) (\x0.\x1. x0 (x0 (x0 x1))) )  -- conversion

    ( (\n.  (\f.(\x. f (f x))) n) (\x0.(\x1. x0 (x0 (x0 x1)))) )  -- Substitution
    ( (\f.(\x. f (f x))) (\x0.(\x1. x0 (x0 (x0 x1)))) )  -- Substitution

    ( ((\x. (\x0.(\x1. x0 (x0 (x0 x1)))) ((\x0.(\x1. x0 (x0 (x0 x1)))) x))) )  -- Substitution

    ( ((\x. (\x0.(\x1. x0 (x0 (x0 x1)))) ((\x2.(\x3. x2 (x2 (x2 x3)))) x))) )  -- conversion

    ( ((\x. ((\x1. ((\x2.(\x3. x2 (x2 (x2 x3)))) x) (((\x2.(\x3. x2 (x2 (x2 x3)))) x)
    (((\x2.(\x3. x2 (x2 (x2 x3)))) x) x1)))) )) )  -- Substitution

    ( ((\x. ((\x1. ((\x2.(\x3. x2 (x2 (x2 x3)))) x) (((\x4.(\x5. x4 (x4 (x4 x5)))) x)
    (((\x6.(\x7. x6 (x6 (x6 x7)))) x) x1)))) )) )  -- conversion

    ( ((\x. ((\x1. (\x3. x (x (x x3))) (((\x4.(\x5. x4 (x4 (x4 x5)))) x)
    (((\x6.(\x7. x6 (x6 (x6 x7)))) x) x1)) )))))  -- Substitution

    (\x. (\x1. (x (x (x (((\x4.(\x5. x4 (x4 (x4 x5)))) x) (((\x6.(\x7. x6 (x6 (x6 x7)))) x)
    x1)) )))))  -- Substitution

    (\x. (\x1. (x (x (x (((\x5. x (x (x x5)))) (((\x6.(\x7. x6 (x6 (x6 x7)))) x)
    x1)) )))))  -- Substitution

    (\x. (\x1. (x (x (x (x (x (x (((\x6.(\x7. x6 (x6 (x6 x7)))) x) x1)))) )))))  -- Substitution

    (\x. (\x1. (x (x (x (x (x (x ((\x7. x (x (x x7))) x1)))) )))))  -- Substitution

    (\x. (\x1. (x (x (x (x (x (x (x (x (x x1)))))))) )))  -- Substitution, final

Algrbra formula:

    f(m,n) = n^m
```

2.7 Week 7 (line 400)

1. REFERENCE, in lines 5-7 and also in lines 18-22 explain for each variable

- Whether it is bound or free
- If it is bound say what the binder and the scope of the variable are

lines 5-7:
evalCBN, and subst are function names declared outside our scope and thus are **free**.
EApp, and EAbs are type variables declared elsewhere and thus are **free** within our scope.
e1, e2, e3, i, and x are placeholders and can be interchanged with another fresh variable at will making them **bound**.
lines 18-22:
subst, and fresh are function names declared outside our scope and thus are **free**.
id, EAbs, and EVar are type variables declared elsewhere and thus are **free** within our scope.
s, id1, e1, f, and e2 are placeholders and can be interchanged with another fresh variable at will making them **bound**.
2. evalCBN part of hw5 using equal sign

```
evalCBN( EApp (\x.x) ((\y.y) a) )
```

```
= EAbs x x → evalCBN( subst x ((\y.y) a) x )

= evalCBN( subst x ((\y.y) a) x )

= evalCBN( EApp (\y. y) a )

= EAbs y y → evalCBN(subst y a y)

= evalCBN(a)

= a
```

3. This item is as the previous one, but for a different lambda term, namely '(..x) y z'

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

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

= (\y'. y) z

= y
```

4. https://hackmd.io/@alexhkurz/BJ7AoGcVK
Consider the listed ARSs

1. $A = \{\}$

$$\boxed{}$$

✓ Terminating
✓ Confluent
✓ UNF's

2. $A = \{a\}$, $R = \{\}$

$$\boxed{a}$$

✓ Terminating
✓ Confluent
✓ UNF's

3. $A = \{a\}$, $R = \{(a, a)\}$

$$\begin{tikzcd} a \arrow[loop above] \end{tikzcd}$$

✗ Terminating
✓ Confluent
✗ UNF's

4. $A = \{a, b, c\}$, $R = \{(a, b), (a, c)\}$

$$\begin{tikzcd} & a \arrow[dl] \arrow[dr] & \\ b & & c \end{tikzcd}$$

✓ Terminating
✗ Confluent
✗ UNF's

5. $A = \{a, b\}$, $R = \{(a, a), (a, b)\}$

$$\begin{tikzcd} a \arrow[loop above] \arrow[d] \\ b \end{tikzcd}$$

✗ Terminating
✓ Confluent
✗ UNF's

6. $A = \{a, b, c\}$, $R = \{(a, b), (b, b), (a, c)\}$

$$\begin{tikzcd} & a \arrow[dl] \arrow[dr] & \\ b \arrow[loop left] & & c \end{tikzcd}$$

✗ Terminating
✗ Confluent
✗ UNF's

7. $A = \{a, b, c\}$, $R = \{(a, b), (b, b), (a, c), (c, c)\}$

$$\begin{tikzcd} & a \arrow[dl] \arrow[dr] & \\ b \arrow[loop left] & & c \arrow[loop left] \end{tikzcd}$$

✗ Terminating
✗ Confluent
✗ UNF's

Try to find an example of an ARS for each of the possible 8 combinations.

Conf.	Term	UNF's	example
T	T	T	$A = \{a, b, c, d\}$, $R = \{(a,b), (a,c), (b,d), (c,d)\}$
T	T	F	$\emptyset$
T	F	T	$A = \{a, b, c, d\}$, $R = \{(a,b), (a,c), (b,d), (c,d), (d,d)\}$
T	F	F	$\emptyset$
F	T	T	$A = \{a, b\}$, $R = \{(a,b)\}$
F	T	F	$A = \{a, b, c\}$, $R = \{(a,b), (a,c)\}$
F	F	T	$\emptyset$
F	F	F	$A = \{a, b\}$, $R = \{(a,b), (b,a)\}$

1.

$$\begin{tikzcd}
& a \arrow[dl] \arrow[dr] & \\
b \arrow[dr] & & c \arrow[dl] \\
& d &
\end{tikzcd}$$

3.

$$\begin{tikzcd}
& a \arrow[dl] \arrow[dr] & \\
b \arrow[dr] & & c \arrow[dl] \\
& d \arrow[loop below] &
\end{tikzcd}$$

5.

$$\begin{tikzcd}
a \arrow[d] \\
b
\end{tikzcd}$$

6.

$$\begin{tikzcd}
& a \arrow[dl] \arrow[dr] & \\
b & & c
\end{tikzcd}$$

8.

$$\begin{tikzcd}
a \arrow[d, bend left] \\
b \arrow[u, bend left]
\end{tikzcd}$$

2.8 Week 8 (line 450)

Rewrite rules are

```
aa -> a
bb -> b
ba -> ab
ab -> ba
```

- Why does the ARS not terminate?

  The ARS can get caught in an infinite loop between 'ab' and 'ba'

- What are the normal forms?

  Normal forms of the ARS are a b

- Can you change the rules so that the new ARS has unique normal forms (but still has the same equivalence relation)?

```
aa -> a
bb -> b
ba -> ab
```

- What do the normal forms mean? Describe the function implemented by the ARS.

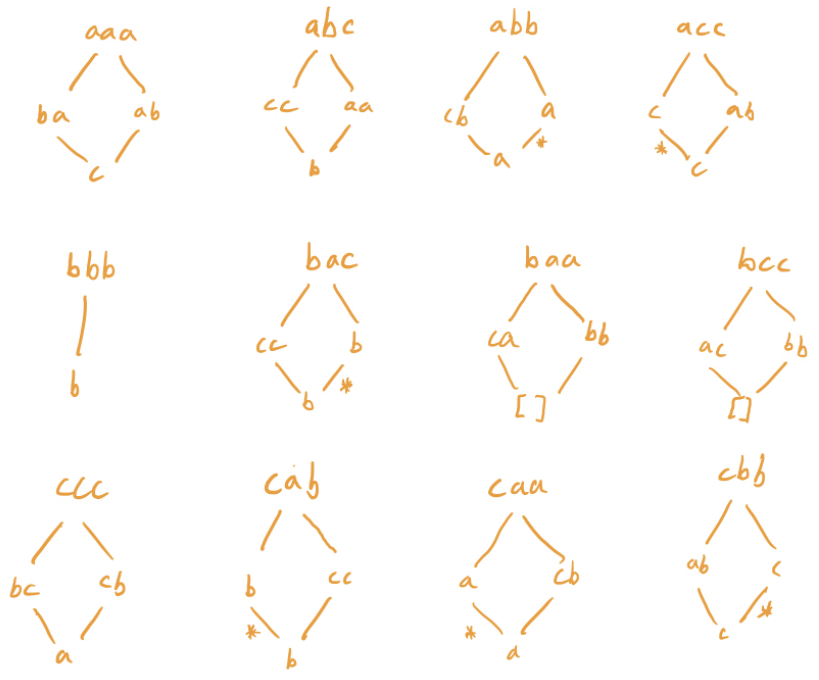
  The normal forms is a reduction determining if the string contains either an a, b, or both. a's are sifted to left, while b's are shifted to right, duplicates are reduced.

2.9 Week 9 (line 470)

consider the ARS:

```
ba -> ab
ab -> ba
ac -> ca
ca -> ac
bc -> cb
cb -> bc

aa -> b
ab -> c
ac ->
bb ->
cb -> a
cc -> b
```

- Normal forms: a, b, c, and [] (empty string)
- We can prove confluence by examining 3 letter combinations:



- By assigning the following equalities:

```
    [] = 0
    a = 1
    b = 2
    c = 3
```

```
we are able to observe an invariance with the mathematical function:

    f(x,y) = (x + y) mod 4
```

2.10 Week 10 (line 520)

```
1.
    let rec f = (\ x . S x) in f 0

    (\f. f 0) (\x. S x)
    (\x. S x) 0
    S 0

2.
    let rec f = (\ x . if x = 0 then 0 else f (minus_one x)) in f (S 0)

    (\f. f (S 0)) (\ x . if x = 0 then 0 else f (minus_one x))
    (\x. if x = 0 then 0 else f (minus_one x)) (S 0)
    if (S 0) = 0 then 0 else f (minus_one (S 0))
    f (minus_one (S 0))
    f (0)

    (\.f f 0) (\ x . if x = 0 then 0 else f (minus_one x))
    (\ x . if x = 0 then 0 else f (minus_one x)) 0
    if 0 = 0 then 0 else f (minus_one 0)
    0

3.
    let rec f = (\ x . if x = 0 then 0 else f (minus_one x)) in f (S S 0)

    (\f. f (S S 0)) (\ x . if x = 0 then 0 else f (minus_one x))
    (\ x. if x = 0 then 0 else f (minus_one x)) (S S 0)
    if (S S 0) = 0 then 0 else f (minus_one (S S 0))
    f (minus_one (S S 0))
    f (S 0)

    (\.f f (S 0)) (\ x . if x = 0 then 0 else f (minus_one x))
    (\ x . if x = 0 then 0 else f (minus_one x)) (S 0)
    if (S 0) = 0 then 0 else f (minus_one (S 0)))
    f (minus_one (S 0)))
    f (0)

    (\.f f 0) (\ x . if x = 0 then 0 else f (minus_one x))
    (\ x . if x = 0 then 0 else f (minus_one x)) 0
    if 0 = 0 then 0 else f (minus_one 0)
    0
```

4.

```
    let rec f = (\ x .
        if x = 0
        then x
        else c (f (minus_one x)))
    in
        f (S 0)

    (\f. f (S 0)) (\x. if x = 0 then x else c (f (minus_one x)))
    (\x. if x = 0 then x else c (f (minus_one x))) (S 0)
    if (S 0) = 0 then (S 0) else c (f (minus_one (S 0)))
    c (f (minus_one (S 0)))
    c (f (0))

    c ((\f. f 0) (\x. if x = 0 then x else c (f (minus_one x))))
    c ((\x. if x = 0 then x else c (f (minus_one x))) 0)
    c (if 0 = 0 then 0 else c (f (minus_one 0)))
    c (0)
```

2.11 Week 11

A domain specific programming language (DSL) for financial contracts.

What if the logic of our justice system was so fine-tuned it could be described in a DSL with precise inputs producing clear outputs? Before that though, we must of course consider legal contracts. Such DSL contracting languages have been proposed and developed mainly in the last 10 years, but none have achieved a legally recognized status. Despite this, trustworthy contracts are still being deployed, and functioning properly without the enforcement of a legal entity. I think my question is: is it possible to create a replacement enforcement entity through DSL's? and to what extent? Can laws of everyday life be determined by smart contracts?

2.12 Week 12

Apply the method of analysis from the lecture to

```
    while (x!=0) do z:=z*y;  x:= x-1 done
```

What is the invariant? Indicate the reasoning steps in which you apply the rules of Hoare logic.

Assuming z is initialized 1, an invariant of $z = y^{n-x}$ is observed. Such equation is a loop invariant and will return the value of z for each step in the looping function.

I derived this mathematical description by constructing a table of execution assuming $z = 1, x = n, y = m$ with arbitrary values for n and m. Additionally a variable T associated with the amount of executed loops in the sequence.

On each loop, x iterates down by 1, while T counts up by 1, thus $T = n - x$. The value of z indicates a function describing y to the power of n - x (x, who's value is iterating down).

3 Project

Smart Contract Programming Semantics

3.1 Specification

For my project I will compare different Smart Contract languages. The languages I have identified are Solidity (Ethereum VM), and Move (Sui VM).

I will implement an identical project in both languages to explore and showcase key features and differences in either systems.

My project will be an auction house smart contract in which a seller will lock an NFT object within the contract, and users will bid under some criteria. The contract should act as an escrow service.

Planned milestones are as follows:

- Research and identify key differences between smart-contracting languages. Brainstorm project idea which incorporates notable differences.
- Build project in both languages and
- Compile report on findings

3.2 Prototype

3.3 Documentation

3.4 Critical Appraisal

. . .

4 Conclusions

Thanks, goodbye.

References

[PL] Programming Languages 2022, Chapman University, 2022.