CPSC-354 Report

Jackson Goldberg
Chapman University

October 30, 2022

Abstract

My Report.

Contents

1 Introduction

Hi there. My name is Jackson Goldberg. I am a senior here at chapman University and I am taking programming languages. I like playing chess. Enjoy reading my report.

2 Homework

2.1 Week 1

I have added my program into this github repo. It is a simple python program that loops infinatly until a and b are equal. I ran it with python 3 and manually input 9 and 33 yeilding the answer 3.

2.2 Week 2

This week we were tasked with creating simple recursive programs in huskall. These are my solutions for all of the assigned functions. These functions can also be found in a huskell file titled "Main.hs".

```
-- Takes a list of char and returns a list of char. This works by assinging the element at odd
    positional values of a list into an empty list using the zip function which is what it
    returns.
select_evens :: [[a]] -> [[a]]
select_evens xs = [x | (x,i) <- zip xs [0..], odd i]

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

I referenced this.

```
-- Same logic as above just assignes based on even possition.
select_odds :: [[a]] -> [[a]]
select_odds xs = [x | (x,i) <- zip xs [0..], even i]

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

```
-- Uses the filter to function to create a list of all matching elements to the ones provided.
   If the length is greater then 0 the its a member.
member :: Int -> [Int] -> IO Bool
member x li= do
  let xs = filter(== x) li
  if length xs == 0
      then
          return (False)
      else do
          return(True)

-- Example:
member 2 [5,2,6] =
1 = filter(== 2) li
if length xs == 0
      else do
        return(True)
True
```

```
-- Uses ++ to concatinate lists.
append :: [Int] -> [Int] -> [Int]
append l1 l2 = l1 ++ l2

--Example:
append [1,2] [3,4,5] =
  [1,2] ++ [3,4,5] =
[1,2,3,4,5]
```

```
-- Recursively assigns element to back of new list, creating a reverse list.
revert :: [Int] -> [Int]
revert [] = []
revert (x:xs) = revert xs ++ [x]

--Example:
revert [1,2,3] =
   [3]: revert[1,2] =
 [3,2]: revert[1] =
 [3,2,1]
```

```
-- Compares two strings using <=.
less_equal :: [Int] -> [Int] -> IO Bool
less_equal l1 l2 = do
    if last l1 <= last l2
        then
            return (True)
        else do
             return (False)
--Example:
less_equal [1,2,3] [2,3,2] =
3 > 2
False
```

2.3 Week 3

This week we were tasked with finishing the hanoi file supplied to us. The full file can be found in Hanoi.txt. Hanoi is used 31 times in the file. You can express this as formula because it doesn't matter the starting blocks since the process will always be the exact same. you will always need to reduce to the highest block so the number is nonconsiquental. There are many cool visuals of this online which show what is looks like using a graphical interpritation. There is a simple break down for however many N you have.

2.4 Week 4

This week we worked with context free grammers and parse trees. Part one can be found here, part 2 can be found here.

2.5 Week 5

This week we are doing a lot of lambda calc stuff with abstract syntax trees.

Using the parser to generate linearized abstract syntax trees:

```
Input:
```

```
\ x . x a

[Abstract Syntax]

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

 Output:
a
```

```
 Input:
 \ x . x a

 [Abstract Syntax]

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

  Output:
 \ x . x a
```

```
 Input:
 \ x . \ y . x a b

 [Abstract Syntax]

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

  Output:
 a
```

```
 Input:
 \ x . \ y . y a b

 [Abstract Syntax]

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

  Output:
 b
```

```
 Input:
 \ x . \ y . x a b c

 [Abstract Syntax]

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

  Output:
 a c
```

```
Input:
\ x . \ y . y a b c

[Abstract Syntax]

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

 Output:
b c
```

```
Input:
\ x . \ y . x a (b c)

[Abstract Syntax]

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

 Output:
a
```

```
Input:
\ x . \ y . y a (b c)

[Abstract Syntax]

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

 Output:
b c
```

```
Input:
\ x . \ y . x (a b) c

[Abstract Syntax]

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

 Output:
a b
```

```
Input:
\ x . \ y . y (a b) c

[Abstract Syntax]

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

```
 Output:
c
```

```
 Input:
 \ x . \ y . x (a b c)

 [Abstract Syntax]

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

 Output:
 \ yx0 . a b c
```

```
 Input:
 \ x . \ y . y (a b c)

 [Abstract Syntax]

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

 Output:
 \ yy0 . yy0
```

Evaluate using pen-and-paper the following expressions:

```
(\x.M) N
N = argument
M = function operation

(\x.x) a = a
\x.x a = \x.x a   cannot be reduced further because there are no parentheses
(\x.\y.x) a b = (\x.(\y.x) a) b
                = (\x.x) a
                = a
(\x.\y.y) a b = (\x.(\y.y) a) b
                = (\y.y) b
                = b
(\x.\y.x) a b c = ((\x.(\y.x) a) b) c
                    = ((\y.a) b) c
                    = (\y.a) b c
                    = a c
(\x.\y.y) a b c = ((\x.(\y.y) a) b) c
                    = ((\y.y) b) c
                    = b c
(\x.\y.x) a (b c) = ((\x.(\y.x)) a) (b c)
                      = (\y.a) (b c)
                      = a
(\x.\y.y) a (b c) = ((\x.(\y.y)) a) (b c)
                      = (\y.y) (b c)
                      = b c
(\x.\y.x) (a b) c = ((\x.(\y.x)) (a b)) c
```

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

All of the hand writen notes can be found Here.

2.6 Week 6

This week for homework we did lambda calc beta reduction on exponentiation. The file containing my solution can be founnd in week6.hs Here (Has .hs for sake of parentheses convienience).

2.7 Week 7

Line 5: e1 and e2 are being bound by the = and the scope is until line 7
Line 6: i and e3 are being bound by the -¿ and the scope goes until line 6
Line 7: e3 is being bound by -¿ and the scope is line 7
Line 8: x is bound by = and the scope is line 8
Line 18: id s id1 e1 are all bound by = and the scope is to the end of line 22
line 20: f is bound by = and the scope goes until the end of line 22
line 21: e2 is bound by = and the scope goes until the end of line 22
The Rest of this weeks homework can be found here.

2.8 Week 8

The ARS does not terminate because lines 3 and 4 are a loop.
a is a normal form of aa and b is a normal form of bb
No because by definition lines 3 and 4 are infinatly reducing into it's self so you would need to modifty the equivilance in order for the ars to achive normal forms. If you removed lines 3 or 4 the ARS would have normal forms.
The normal forms provide termination, this is essentailly like a base case. The two functions would be:
F(aa) = a;
F(bb) = b;

2.9 Week 9

1) The deadlines for my project:
a. The first deadline November 20th, I will have a writen guild on the rust language breaking down the language.
b. The second deadline Novemeber 27th, I will have a scripted server in rust to show the backend capabilities.
c. The third deadline December 4th, I will have both a Rust server and a C server and compare and contrast the differences.
2) The assignment
This week for our assignment we were asked to evaluate the ARS found in "Homework 9" on canvas. We were tasked with finding the function which corsponds with the set off rules very similar to a puzzle. One of the first things I noticed is that there were only 3 normal forms: the empty string (""), a, and b. I also noticed there were not equivilance cases, such that a can be equal to b and so on. With that in mind I set a=1, b=2, and c=3 with the empty string equal to 0 (""=0). From this I could see that the only normal

forms were 1 and 2. Given that I knew this ARS couldn't be calculating something like addition, subtraction, or multiplication. I first attempted division. However there was not a case in which the answer could be zero with natural numbers. I eventually landed on Mod being the most likely answer since given the normal forms it is a function that given the right donominator can yeild 0, 1, and 2 as the only answers. I then checked it against many examples of combinations of a, b, and c. I found that it was not consistant if the exact string was converted to a number but I did figure out that if you add the numbers together instead of directly translating them that you do end up getting only the remainers 0, 1, and 2 when you divide by 4. Therefor my answer to the puzzle is that this ARS is the ruleset for SUM of any combination (a,b,c) MOD 4.

3 Project

Introductory remarks ... Over the summer I had an internship and I primarily worked using rust. I didn't get to learn it very well but the little I did learn got me really interested in it. For my project I would like to do a deep dive into rust and look at the advantages it has over a similar language like C then show off a project with it.

3.1 Specification

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.