CPSC-354 Report

Jackson Goldberg
Chapman University

December 19, 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

Assignment: The aim of this homework is to familiarize yourself with LaTeX. Add your program (one is enough) to report.tex and explain in detail how it executes on a sample input such as gcd(9,33)

I have added my program into this github repo. It can be found here.
It is a simple python program that loops infinatly until subtracting the first and second values from their counterpart until they are equal.
Once they are equal then the program prints the results and terminates. You can run in python and it will take in two numbers from the command line.
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.
The equation I came up with is H(n) = $(2^n - 1) - 1$

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)
```

ASTs 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 (rightarrow) and the scope goes until line 6
Line 7: e3 is being bound by (rightarrow) 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

Q1. Why does the ARS not terminate?
The ARS does not terminate because ba $\rightarrow$ *abandab* $\rightarrow$ *badonotstopreducing*.
Q2. What are the normal forms?
$The normal forms are : a, b, []because they are unable to be reduced further.$
Q3. Can you change the rules so that the new ARS has unique normal forms (but still has the same equiva
$For this ARS to have unique normal forms the ARS would need to terminate. Termination is not possible right now with the given se$
Q4. What do the normal forms mean? Describe the function implemented by the ARS?
$The normal forms in this case represent something that is ireducable as it they are not able to be reduced further. This function takes$

2.9 Week 9

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 4 normal forms: the empty string (""), a, b, and c. With that in mind I set a=1, b=2, and c=3 with the empty string equal to 0 (""=0). 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.

2.10 Week 10

The homework can be found here.

2.11 Week 11

My Question:
I asked this question in class but since I saw that it wasn't added to the list of questions I thought I would still post it:

"What can a DSL not do?"

Essentially I am asking what are the limitations of a DSL since they are inherently going to be more restricted then an object oriented programming language given their nature. Is this solution given those limitations something that actually provides more value then making something like a parser which can do something similar? I was really happy with this question and I thought I would share it.
My response to Meghna:
So I am not sure if I have the correct answer but I can offer maybe one line of thinking. I think that to your first point it all depends on the implementation. Contracts are tricking pieces of legally binding paper and the dynamic nature of them might prove to be challenging to cover for the implementers of the DSL. To your second question I think of the European option as a separate entity from the American option and it makes more sense then them being related for the sake of uniqueness.
My response to Andriana:
This is where I think it might speed up the process because once you have something that can be typed that has a strict set of rules the only thing that fallows is memorization which I think is a good trade off for the efficiency that one might acquire by picking up these skills.

2.12 Week 12

What is the invariant? Indicate the reasoning steps in which you apply the rules of Hoare Logic.
while (x!=0) do z:=z*y; x:= x-1 done
We can write out our logic like:
x¿=0while (x!=0) do z:=z*y; x:= x-1 done
Precondition:
x = m
y = x
z = n
We can also initialize a veriable t to 0 and use it to keep track of loop exuction.
With that we get:

t x y z
0 100 2 1
1 99 2 2
2 98 2 4
... ...
99 1 2 $2^9$9
100022^{1}00
$t + x = 100 - >$
$t = 100 - x$
$z = y^t$
$The invariant z = t * y^(m - x)$

3 Project

Introductory remarks ... Over the course of this last summer I got the opertunity to program in rust. I wasn't able to go really in depth but the functionality of the languge really caught my attention. Rust has a lot of really interesting features making it feel like a modern version of C or C++ to me. What I set out to do with my project was show off the differences that sets rust apart from C and C++ in attempts to convince the reader that there are use cases where programming in rust is more valuable.

3.1 Specification

I decided for my project I am going to show off two main use cases for Rust and why you would want to use it instead of C. The first case will be the memory protection. The second case will be abstraction and readablility of Rust compared to C.

3.2 Prototype

Part 0:
A full breakdown of the differences between C and Rust can be found in this report.
Part 1:
All of my code for the fallowing demonstrations will be linked and it located in the local repository.

Memory management in programing is something very important since without it you will end up with low and inefficient programs. One example of a way memory is handled is with pointers and unsued memory. When programming in c their is no memory protection with unused memory on the heap. The way Rust handles it is by having a owner system to memory.

I have included two files, memorytest.c and memorytest.rs these files both show case a large difference between c and rust. In them I made the code as identitical as possible while still respecting their syntax.

Idea summary: show how memory assignment differs in c and rust back to back.

C Code:

```
#include <stdio.h>
struct Example{
    int a_integer;
    char a_string[3];
};

int main (){
  struct Example someData;
  struct Example someOtherData;
```

```
    someData.a_integer = 1;
    someData.a_string[0] = 'h';
    someData.a_string[1] = 'i';

    someOtherData.a_integer = 2;
    someOtherData.a_string[0] = 'B';
    someOtherData.a_string[1] = 'y';
    someOtherData.a_string[2] = 'e';

    someData = someOtherData;

    printf("%c %c %c", someData.a_string[0], someData.a_string[1], someData.a_string[2]);
    printf("%i", someData.a_integer);
  };
```

C Code Output:

```
B y e2
```

What this means:
This test case is showing that the you are able to freely reassign existing variables to point to different memory locations (in this example a struct). Since we are seeing that it returned the data from someOtherData the main question is what happened to the data held by someData? Well since it was not properly deleted that memory that someData was pointing to is left on the heap. This is a really big deal because if we implementing a system where we need to reallocate variables dyncamicly a security risk could be doing too many reassignments with no clean up. If that were to happen it could compromise the hard drive and cause all kinds of issues. The solution to this was to implement garbage collectors.

What happens when we attempt to do the same operation in Rust?

Rust Code:

```
  struct Example{
    a_integer: i32,
    a_string: String,
}
impl Example{
    fn toString(&self) -> String{
        format!("{} - {}", self.a_integer, self.a_string)
    }
    fn println(&self){
        println!("{}",self.toString());
    }
}

fn main(){

    let mut someData = Example{
        a_integer: 1,
        a_string: String::from("hi"),
    };
    let someOtherData = Example{
        a_integer: 2,
        a_string: String::from("bye"),
```

```
    };
    someData.println();
    someData = someOtherData;
    someData.println();
    someOtherData.println();
}
```

Rust Code Output:

```
  error[E0382]: borrow of moved value: ‘someOtherData‘
   |
20 |     let mut someOtherData = Example{
   |         ----------------- move occurs because ‘someOtherData‘ has type ‘Example‘, which does
       not implement the ‘Copy‘ trait
...
25 |     someData = someOtherData;
   |                ------------- value moved here
26 |     someData.println();
27 |     someOtherData.println();
   |     ^^^^^^^^^^^^^^^^^^^^^^^ value borrowed here after move
```

What this is showing:
This is an example of Rust's ownership system. In Rust no two references are allowed to point to the same location of memory due to memory safety. Only one reference can point to one locationn in memory. Rust is implemented this way to avoid the problems defined by the C example. The downside is that you are not able to share memory locations dyncamicly however you are able to ensure that there are no loose chunks of memory on the heap. Due to this implementation Rust also does not require a garbage collection system since these errors are found at compile time.

Part 2:

In this part I have included two simple http servers. I did not write these servers and they are not origional code. Their github links can be found in the documentation section in addition to how to run and use them.

Rust Server

C Server

What I want to highly about this use case is the readablility, abstraction, and length of the Rust server compared to the C server. Both of the servers when you run them give you a very similar output to your local host. The main difference however isn't with the end result but with the programs themselves. The rust server consists of less then 50 lines of code in total while still being very reable and manageable. The C server on the otherhand has an extensive code body across multiple files. In this specific case the rust server is a lot more practical for a use case due to it's abstraction. Rust is also able to have this level of abstraction without sacrificing any proformance.

3.3 Documentation

All Rust programs were compiled using rustc
All C programs were compiled using gcc
To run the rust files:
rustc (name of file).rs
./(name of file).exe

To run the C files:
gcc (name of file).c
./a.out (for memory test)
./server (for sample server)

The source code for the rust server can be found here.

The source code for the C server can be found here.

3.4 Critical Appraisal

The reason I decided on this project was because Rust was a language that really interested me that I didn't know a lot about. I had heard it get compared to C and C++ very often as an alternative. I wanted to know what really differentiated it from its competitors since even though it is a mature language it is really young compared to C and C++. I thought that it related a lot to the class as well since we have been talking able and comparing different programs all semester. Especially something like memory management isn't something that we ecspiletly covered this semester but I think it's very relevant because even at a low level it effects the efficiency of a program significantly. One difficult thing I ran into while doing this project was finding any kind of code comparing the two languages with any type of benchmarking tests. I had to write all of the comparison code and be as possible to each language as I could. If I were to go back and do it again I would expand on my use cases and showcase something more practical, like how two different larger programs preform differently and why the memory protection in Rust is so important.

...

4 Conclusions

I really loved how this course felt like a practical application of discrete math. I know we talked a little about haskel in discrete math but it was really cool getting to use it and write basic programs in it. Another thing I really enjoyed was stripping away all of the layers of abstraction and getting to the core about what is actually happening and physically seeing how division is more complicated than multiplication with all of the different recursive calls. Another thing I really enjoyed was drawing out all of the abstract syntax trees and seeing the different processes and states and talking about different forms of functional programs and how they took advantage of this. I think if there was anywhere in this course where I could be critical I think it would be with the content of the course after week 10 or so and also the final project. After around week 10 I felt like the lectures became a little unorganized and sporadic and didn't follow as tight of focus like the lectures did in previous weeks. That made it hard for me to grasp the fundamentals of what we were learning at the time. For the final project I really wish that expectations were a bit more clear. It seemed very open ended which is great because it allows a lot of flexibility in the project but for me, since there wasn't a rubric, I always was concerned that I was not doing enough or that my work wasn't good enough quality. I think one way to improve this might be by posting some example projects so it might be a little easier to understand the scope. To conclude I was really fascinated by this course. This course really forced me to think in ways that I am not used to and that was something I really enjoyed. It also got me accustomed to a lot of different languages and introduced me to lambda calculus which has already come up in different facets since learning about it. Thank you for your passion for this class and I wish I had the time to take compilers because I find all of this very fascinating.

References

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