CPSC-354 Report

Rohm Tandon
Chapman University

07/01/2024

Abstract

This report contains assignments throughout the fall 2024 semester and is intended for the purpose of documenting my work and showing my progress in CPSC 354 - Programming Languages, taught by Jonathan Weinberg.

Contents

1 Introduction

This report, prepared for CPSC 354 - Programming Languages at Chapman University, is a comprehensive account of my academic voyage over the semester. It includes a detailed compilation of my notes, homework solutions, and critical reflections on the coursework. This report serves as a bridge between the theoretical knowledge imparted in lectures and the practical skills essential for future pursuits in both graduate studies and the software industry.

2 Week by Week

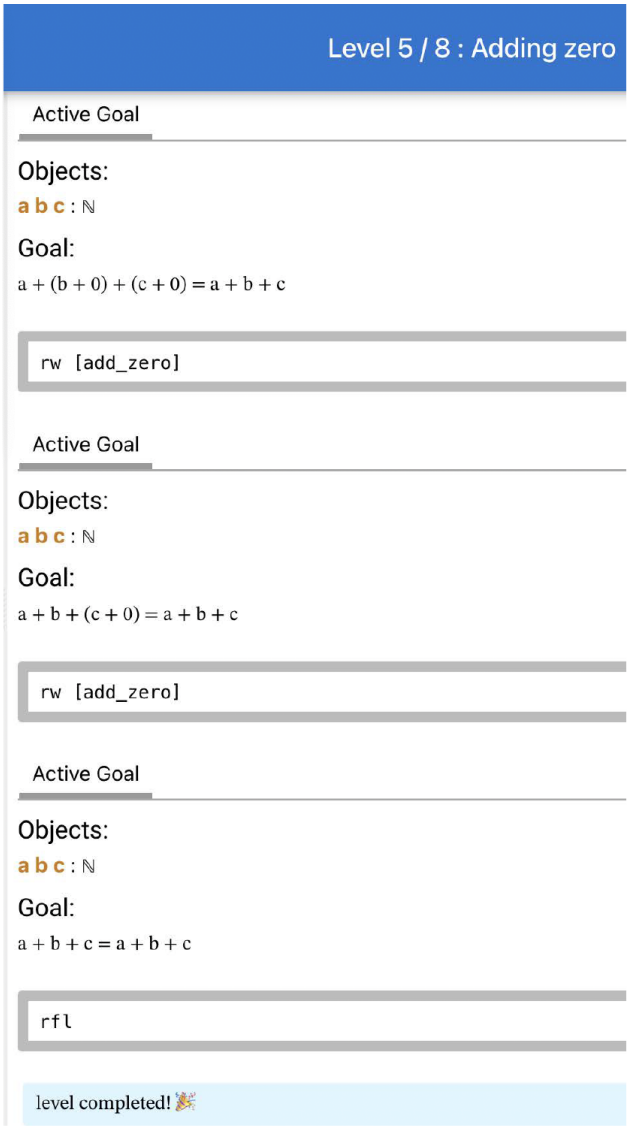
2.1 Week 1

Notes

In week 1 we learnt about Lean as a programming language and its correlation to discrete math. We also learnt about other proof assistants. We then shifted our focus to the NNG tutorial world as you can see below.

Homework

Tutorial world



Level 5:

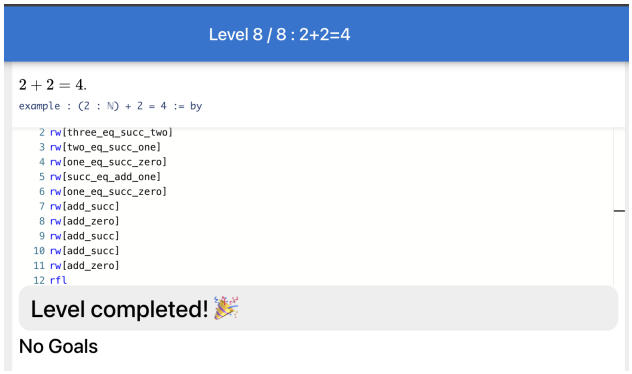
Discrete math's lemmas tell us that anything added to 0 will give the result of that number itself. So; A+0=A. Using this we can bring the left hand side down to a+b+c. From here we can use the property of reflexivity to show that both sides are equal, hence solving the puzzle.

Level 6:


Level 6 / 8 : Precision rewriting
Objects:
a b c : ℕ
Goal:
a + (b + 0) + (c + 0) = a + b + c
rw [add_zero c]
Active Goal
Objects:
a b c : ℕ
Goal:
a + (b + 0) + c = a + b + c
rw [add_zero b]
Active Goal
Objects:
a b c : ℕ
Goal:
a + b + c = a + b + c
rfl
level completed! 🎉

Level 7:

Level 8:



2.2 Week 2

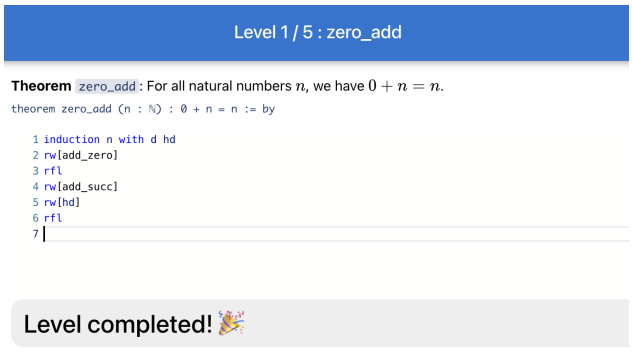
Notes

In week 2 we learnt about recursion and its application in other problems such as the Towers of Hanoi game we played. We also learnt about its various benefits such as breaking down complexity of problems and being more concise.

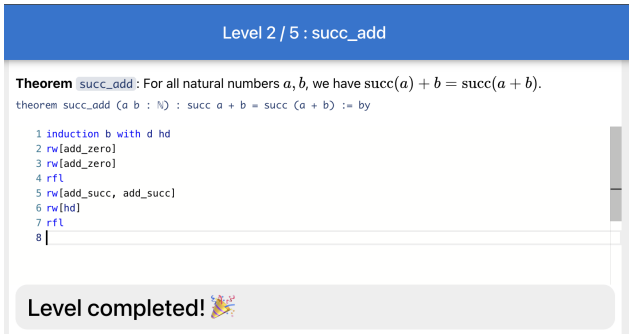
Homework

Addition world

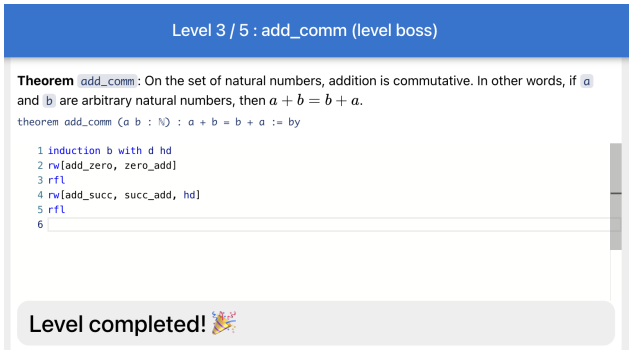
Level 1:



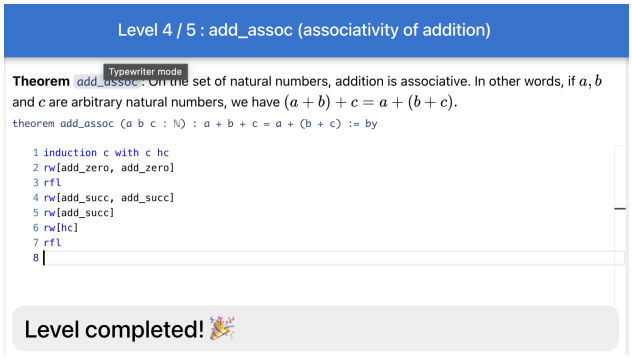
Level 2:



Level 3:

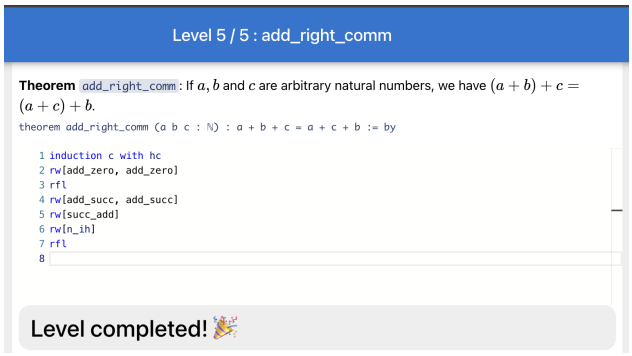


Level 4:



Using induction on c we can initially create an easier medium to use reflexivity to solve for a+b. Then solving the other side we just use the mathematical definiton of a successor function, until we can use the induction again to get the equation to the point where we can use reflexivity to prove it. This is a clear example of mathematical prrofs by induction.

Level 5:



Once again this is proof by mathematical induction similar to the previous one. This time we add the zeroes and then use reflexivity for the first part of the proof. Then to prove the second part we use the successor function until bringing back the induction we used like the previous question. Then to complete the proof we use reflexivity again.

-

Discord Question: Since recursion has so many benefits and also breaks down the complexity of problems, why aren't we taught to use it as our primary method? In other words, why isn't it the first method of problem solving we're taught?

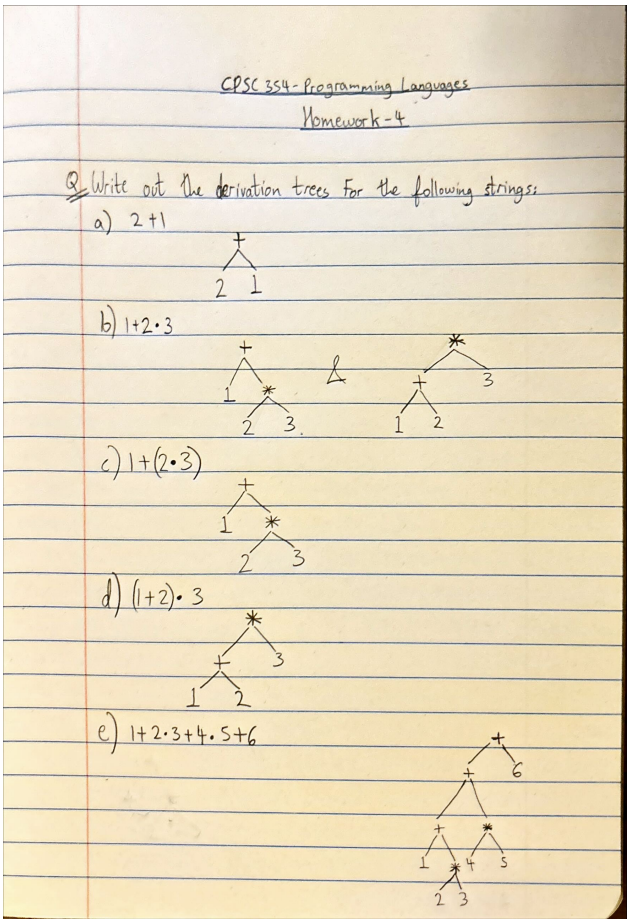
2.3 Week 3

Notes

In week 3 we spoke about recursion further, and focused on our calculators in python. Eventually connecting the dots for our first Assignment that was due at the same time as Homework 4, where I used recursion in my python calculator in order to get it to function as efficiently as possible. We also discussed parsing, and derivation trees which is what we practiced in Homework 4.

Homework

For homework 4 we did some practice on derivation trees for the strings you can see in the handwritten work below, along with the respective answers;



Discord Question: Why do programming languages need different types of parsers, and how does this choice impact the way a computer understands the code?

2.4 Week 4

Notes

In week 4 we spoke about parsing and trees further which is what we practiced in last weeks Homework. But now we delved into the more notation heavy side of it. This weeks homework contained the lean logic game that helped us put some of that into practice.

Homework

For homework 5 we completed the Lean logic game tutorial world. I have provided the answers to the same below.

1. **Exhibit evidence that you're planning a party.**

   Level 1:

   ```
   example (P : Prop) (todo_list : P) : P := by
   ```

   Solution:

```
  exact todo_list
```

Level 2:

```
  example (P S : Prop) (p : P) (s : S) : P  S := by
```

Solution:

```
  exact and.intro p s
```

Level 3:

```
  example (A I O U : Prop) (a : A) (i : I) (o : O) (u : U) : (A  I)  O  U := by
```

Solution:

```
  have ou := and_intro o u
  have ai := and_intro a i
  exact and_intro ai ou
```

Level 4:

```
  example (P S : Prop)(vm: P  S) : P := by
```

Solution:

```
  exact vm.left
```

Level 5:

```
  example (P Q : Prop)(h: P  Q) : Q := by
```

Solution:

```
  exact h.right
```

Level 6:

```
  example (A I O U : Prop)(h1 : A  I)(h2 : O  U) : A  U := by
```

Solution:

```
  exact and_intro h1.left h2.right
```

Level 7:

```
  example (C L : Prop)(h: (L  (((L  C)  L)  L  L  L))  (L  L)  L) : C := by
```

Solution:

```
have a:= h.left
have b:= a.right
have c:= b.left
have d:= c.left
exact d.right
```

Level 8:

```
example (A C I O P S U : Prop)(h: ((P  S)  A)  ¬I  (C  ¬O)  ¬U) : A  C  P  S := by
```

Solution:

```
have a:=h.left
have fin:=a.left
have a:=a.right
have r:=h.right
have t:=r.right
have k:= t.left
have c:=k.left
have e:= and_intro c fin
exact and_intro a e


Explanation for Solution 8:

We ultimately need A  C  P  S, which means that we need to retrieve A, C, P, and S from the origi

(1) Seeing that A, P and S are on the left side of h-> extract h.left to get (P  S)  A.
(2) Seeing that (P  S) i already clearly made on the left side we can store (P  S) together as 'f
(3) Now from the same expression we can extract the right side to retrieve and store A as 'a'.
(4) Now that we have A, P and S stored and easily accessible, we need C from the right side of h.
(5) After storing h.right in another expression, 'r', we take the right side of r to further narr
(6) Now we take the left side of that expression for the same purpose, store it as k.
(7) Finally we can retrieve C by storing the k.left as 'c'.
(8) Now to finally present our solution we can start by using and_intro on c and fin, storing tha
(9) To complete our solution we use exact and_intro on a and e.
```

Discord Question: How does the choice of a parsing technique (e.g., top-down vs. bottom-up) impact the efficiency and clarity of the resulting derivation tree? In other words, when should I choose between the two approaches so that my chosen approach is significantly more beneficial than the other?

2.5 Week 5

Notes

In week 5 we spoke about type theory in programming languages. Moreover, we spoke about proposition types, constructive logic, and functional programming. We also furthered our understanding of the lean logic that helped me understand some parts of the homework as well.

Homework

For homework 6 we completed the Lean logic game on implication. I have provided the answers to the same below.

1. **Exhibit evidence that cake will be delivered to the party**

   Level 1:

   ```
   example (P C: Prop)(p: P)(bakery_service : P → C) : C := by
   ```

   Solution:

   ```
   exact(bakery_service p)
   ```

   Level 2:

   ```
   example (C: Prop) : C → C := by
   ```

   Solution:

   ```
   exact  h : C => h
   ```

   Level 3:

   ```
   example (I S: Prop) : I  S → S  I := by
   ```

   Solution:

   ```
   exact  h : I  S => and_intro (and_right h) h.left
   ```

   Level 4:

   ```
   example (C A S: Prop) (h1 : C → A) (h2 : A → S) : C → S := by
   ```

   Solution:

   ```
     exact  h : C => h2 (h1 h)
   ```

   Level 5:

   ```
   example (P Q R S T U: Prop) (p : P) (h1 : P → Q) (h2 : Q → R) (h3 : Q → T) (h4 : S → T) (h5 : T -
   ```

   Solution:

   ```
   exact h5 ( h3 (  h1 p))
   ```

   Level 6:

   ```
   example (C D S: Prop) (h : C  D → S) : C → D → S := by
   ```

   Solution:

```
exact fun f : C => fun b : D => h ⟨f, b⟩
```

Level 7:

```
example (C D S: Prop) (h : C → D → S) : C  D → S := by
```

Solution:

```
exact fun f : C  D => h f.left f.right
```

Level 8:

```
example (C D S : Prop) (h : (S → C)  (S → D)) : S → C  D := by
```

Solution:

```
exact fun s : S => and_intro (h.left s) (  h.right s)
```

Level 9:

```
example (R S : Prop) : R → (S → R)  (¬S → R) := by
```

Solution:

```
exact fun r : R => and_intro (fun s : S => r) (  fun ss: ¬S => r)
```

Discord Question: How can implication be used in functional programming, and more so, why is it not as seemingly embedded in some of the popular languages we use today?

2.6 Week 6

Notes

In week 6 we spoke about Lambda Reduction and practiced a few questions to improve our understanding of the same.

Homework

For homework 7 we discussed some theory related to capture avoiding substitution by reducing a lambda term. Then we read and discussed Church numerals.

1. **The purpose of this hw is to practice capture avoiding substitution.**

   Question 1:

   Solution:

   ```
   - (\n.(\f.\x.f(fx))n)(\f.\x.f(f(fx)))
   - (\f.\x.f(fx))(\f.\x.f(f(fx)))
   - \x.((\f.\x.f(f(fx)))(f(fx)))

   Capture avoiding substitution is the practice of performing substitutions in lambda calculus, whi
   This yields the result: \x.((\f.\x.f(f(fx)))((\f.\x.f(f(fx)))x))
   ```

Question 2:

Solution:

```
Given \m.\n.mn, let's interpret it in terms of Church numerals.
This function takes two Church numerals, m and n.
This function implements addition for Church numerals. It combines the number of applications o
So this function represents the addition function in Church numerals.
```

Discord Question: When performing capture-avoiding substitution, we often rename bound variables to avoid conflicts. How might this affect the efficiency or complexity of the substitution operation?

3 Lessons from the Assignments

4 Conclusion

References

[BLA] Author, Title, Publisher, Year.