CPSC-354 Report

Chaz Gillette

September 8, 2024

Contents

Week 1: Introduction to Lean and Natural Number Game Tutorial

In week one, we worked with Lean and reviewed our discrete math knowledge in the Number Game Tutorial. Problems and solutions are listed below.

Homework Solutions: Week 1

Level 5

```
a + (b + 0) + (c + 0) = a + b + c.
   rw [add_zero]
   rw [add_zero]
   rfl
```

Level 6

```
a + (b + 0) + (c + 0) = a + b + c.
   rw [add_zero c]
   rw [add_zero b]
   rfl
```

Level 7: succ_eq_add_one Theorem

```
Theorem succ_eq_add_one:  For all natural numbers a, we have succ(a)
= a + 1
   rw [one_eq_succ_zero]
   rw [add_succ]
   rw [add_zero]
   rfl
```

Level 8: 2 + 2 = 4

```
2 + 2 = 4.
   nth_rewrite 2 [two_eq_succ_one] -- only change the second ‘2‘ to
‘succ 1‘.
   rw [add_succ]
   rw [one_eq_succ_zero]
   rw [add_succ, add_zero] -- two rewrites at once
   rw [\three_eq_succ_two] -- change ‘succ 2‘ to ‘3‘
   rw [\four_eq_succ_three]
   rfl
```

Detailed Explanation: Level 7 Proof

I chose to explain the proof for level seven because this is where we make the breakthrough with addition. Our goal is to prove that the successor of a is equal to $a + 1$. So, in step one, we want to rewrite one as the successor of zero. That gives us succ $n = n +$ succ 0. The next step is ‘add_succ‘ so that succ $n =$ succ $(n + 0)$. After that, we can remove the zero by ‘rw [add_zero]‘, which will leave us with succ $n =$ succ n, which is thus proven true with the reflexive property.

Lessons from the Assignments

Lesson from Week 1

Week one was our review and introduction to the math side of what we'll be learning this semester. We started by revisiting the basic rules of discrete math. This meant getting back into the flow of writing out our proofs using the rules that we have access to. With only natural numbers to start, we started looking at successors again and eventually proving our way toward addition.

For me, this was a needed refresher because it's been a moment since I took discrete math, and I'm unfamiliar with writing my proofs as code, which is a learning curve for me. I'm a very pen-to-paper mathematician, so thinking about math at the same time I'm trying to recall syntax for code is a challenge for me. That said, we went through eight levels of proofs, and I was able to begin to get the hang of it.

I'm looking forward to bridging the gap between my math knowledge and how I involve it when I code. Sometimes I feel like I have the education to understand the concepts, but I struggle to apply them when I'm coding. The speed in which I type out code is not as quick as how I think about what I'd like to apply. This first assignment was a nice intro to opening my eyes as to what it might look like to get faster at that and also write technical reports in a coding environment as well. By shifting everything I do—the code, the math, the reporting—into an IDE, I know that I'll be able to get more comfortable working in that environment.

Week 2: Finishing the NNG Addition World

In week two, we focused on completing the Natural Number Game (NNG) Addition World, which helped solidify our understanding of addition in Lean. The problems and solutions for Levels 1-5 are listed below.

Homework Solutions: Week 2

Level 1: zero_add

```
theorem zero_add (n :  nat) :  0 + n = n := by
   induction n with d hd
   rw [add_zero]
   rfl
   rw [add_succ]
   rw [hd]
   rfl
```

Level 2: succ_add

```
theorem succ_add (a b :  nat) :  succ a + b = succ (a + b) := by
   induction b with d hd
   rw [add_zero]
   rw [add_zero]
   rfl
   rw [add_succ]
   rw [hd]
   rw [add_succ]
   rfl
```

Level 3: add_comm

```
theorem add_comm (a b :  nat) :  a + b = b + a := by
   induction b with d hd
   rw [add_zero]
   rw [zero_add]
   rfl
   rw [add_succ]
   rw [hd]
   rw [succ_add]
   rfl
```

Level 4: add_assoc

```
theorem add_assoc (a b c :  nat) :  a + b + c = a + (b + c) := by
   induction c with d hd
   rw [add_zero]
```

```
  rw [add_zero]
  rfl
  rw [add_succ]
  rw [hd]
  rw [add_comm]
  rw [add_succ]
  rw [add_succ]
  rw [add_comm]
  rfl
```

Level 5: add_right_comm

```
theorem add_right_comm (a b c :  nat) :  a + b + c = a + c + b := by
  induction c with d hd
  rw [add_zero]
  rw [add_zero]
  rfl
  rw [add_succ]
  rw [hd]
  rw [add_comm]
  rw [add_succ]
  rw [add_comm]
  rw [succ_add]
  rfl
```

Mathematical Proof for Level 5: add_right_comm

The goal is to prove the right commutativity of addition, meaning for all natural numbers a, b, and c, the equation $a + b + c = a + c + b$ holds. This is done by using induction on c.

Base Case: When $c = 0$, we need to prove:

$$a + b + 0 = a + 0 + b$$

Using the identity property of addition, we know that $a + 0 = a$ and $0 + b = b$. Thus, both sides simplify to:

$$a + b = a + b$$

which is clearly true.

Inductive Step: Assume that the right commutativity holds for some c, i.e.:

$$a + b + c = a + c + b$$

Now, we must show that the commutativity holds for $\text{succ}(c)$, i.e., that:

$$a + b + \text{succ}(c) = a + \text{succ}(c) + b$$

By the definition of the successor function and the properties of addition, we can rewrite the left-hand side:

$$a + b + \text{succ}(c) = \text{succ}(a + b + c)$$

By the inductive hypothesis, we know that $a+b+c = a+c+b$, so we substitute this in:

$$\text{succ}(a + c + b) = a + \text{succ}(c) + b$$

which completes the inductive step. Therefore, by the principle of induction, we conclude that for all natural numbers a, b, and c:

$$a + b + c = a + c + b$$

□

Detailed Explanation: Level 5 Proof (add_right_comm)

In this proof the goal is to show that for all natural numbers a, b, and c, the equation $a + b + c = a + c + b$ holds. This is commutativity of addition. I attempted the proof by induction on c.

1. Base case: When $c = 0$, the goal is to prove $a + b + 0 = a + 0 + b$. Using the definition of addition, $a + 0 = a$, so both sides reduce to $a + b$. Then we rewrite tactic 'rw [add_zero]' and 'rfl' confirms it.

2. Inductive step: We assume that $a+b+c = a+c+b$ holds for some c, and we must prove $a+b+\text{succ}(c) = a+\text{succ}(c)+b$. First we rewrite the addition of the successor using the 'add_succ' rule. Then by applying the inductive hypothesis and the commutativity of addition, we can transform both sides to eventually match, proving the equality.

Lessons from the Assignments

Lesson from Week 2

In week two we dived into the realtionship between mathematical proofs and lean proofs. We saw that the code in an of itself is a proof and in each step we are just translating into a language the lean will understand. Additionally, we spent time looking at how to prove things recursively. Looking at functions that call themselves as a solution. This requires a base case with a simple solution, like moving a single ring to the right tower, and the replace that simple solution with n so that we can continue to reduce down to our basecase and solve for any number of rings up to infinity. This is what we do in our unductive proofs, we have our base case, then our hypothesis for n + 1 and prove for all cases.

Conclusion

In week one, we reviewed our discrete math knowledge and began coding proofs. Week two we saw the relationship between mathematical proofs and lean code proofs, and then we began to solve problems recursively.