

CPSC-354 Report

Brandon Hughes
Chapman University

September 10, 2024

Abstract

Contents

1	Introduction	1
2	Week by Week	1
2.1	Week 1	1
2.1.1	Notes	1
2.1.2	Homework	1
2.1.3	Comments and Questions	3
2.2	Week 2	3
2.2.1	Notes	3
2.2.2	Homework	4
2.2.3	Comments and Questions	6
2.3	6
3	Lessons from the Assignments	6
4	Conclusion	6

1 Introduction

2 Week by Week

2.1 Week 1

2.1.1 Notes

During this week, there was a review of Git and being introduced to Latex and Lean. Some helpful commands include git add, commit, status, and push. Through the website "<https://sudorealm.com/blog/how-to-write-latex-documents-with-visual-studio-code-on-mac>", we set up latex to be able to complete the weekly report.

2.1.2 Homework

1. Finish the Natural Number Game Tutorial World.
 - a) Show the completed work for levels 5 through 8.

b) For one level, explain in detail how the Lean proof is related to its corresponding proof in mathematics.

1a. Show the completed work for levels 5 through 8.

Level 5: Prove that $a+(b+0)+(c+0)=a+b+c$.

```
rw[add_zero]
rw[add_zero]
rfl
```

Level 6: Prove that $a+(b+0)+(c+0)=a+b+c$.

```
repeat rw[add_zero]
rfl
```

Level 7: Prove that for all natural numbers a , we have $\text{succ}(a)=a+1$.

```
rw[one_eq_succ_zero]
rw[add_succ]
rw[add_zero]
rfl
```

Level 8: Prove that $2+2=4$.

```
repeat rw[four_eq_succ_three, three_eq_succ_two, two_eq_succ_one, one_eq_succ_zero]
repeat rw[add_succ, add_succ, add_zero]
rfl
```

1b. For one level, explain in detail how the Lean proof is related to its corresponding proof in mathematics.
For level 7, we had to prove the theorem of the $\text{succ}(n)$ is also equal to $n+1$.

Lean Proof:

```
Start: succ(n) = n + 1
1: rw[one_eq_succ_zero]
Result: succ n = n + succ 0
2: rw[add_succ]
Result: succ n = succ (n + 0)
3: rw[add_zero]
End: succ n = succ n
```

Thus, proving reflexivity.

Proof by Mathematics: By using the induction we are able to prove the theorem of the $\text{succ}(a)$ is equal to $a+1$.

Base Case:

Consider $n = 0$,

$S(0) = 0 + 1$

$0 + 1 = 1$

Thus, $0 + 1 = S(0)$, which holds true.

Inductive hypothesis:

Assume for some natural number k that $k + 1 = S(k)$.

Inductive Step:

We need to show that $k + 1 + 1 = S(k + 1)$.

$(k + 1) + 1 = S(k + 1)$, by adding parenthesis

$S(k) + 1 = S(k + 1)$, by using the inductive hypothesis.

$S(k + 1) = S(k + 1)$, by using addition of successors.

Thus, proving reflexivity.

Through these steps, we can see that the end goal of proving reflexivity on both the Lean proof and its corresponding proof in mathematics. The similarities come from the lean proof and the inductive step however, as they are similar steps in being able to prove the theorem. The lean proof is more straight forward because instead of proving it through a basis, inductive hypothesis, and inductive step, you only have to prove it through rewriting the equation so that both sides are equal. Which is done through the inductive step of the mathematics proof.

2.1.3 Comments and Questions

When looking at Formal Systems from the textbook, we are given this example of an impossible puzzle to solve. The MU problem, where you are given a set of rules and have to obtain MU from MI, however, its impossible because you can never end up without having an odd number of I's inside the string. When it comes to Formal systems, and solving them a lot of times people will look towards actually doing compared to trying to assess the logic behind this however computers, mostly AI, generally start at the logic. How might combining human intuition and AI's logical reasoning lead to more effective problem-solving strategies? Would we be able to solve problems quicker, or would AI's logical reasoning overtake the human trial and error method?

2.2 Week 2

2.2.1 Notes

During this week, we learned in class that both Math and Lean can be seen as languages. Math is a specification language while Lean is a programming language. A specification language is used to define the requirements and properties of a system. A Math proof can be written into Lean proof very easily since they use and follow the same rules when it comes to solving theorems and problems. The difference between the two proofs however is that in Math we typically reason forwards from the problem to the answer, while in Lean we reason backwards from the answer to the problem. We could also do it the opposite way but it would become more challenging. Another idea that we learned in class is that a recursive data type, which could also be called an algebraic data type, and induction are of similar processes as you define what a number is inside of a number. An example of recursion can be seen in the Tower of Hanoi as you are solving the previous tree in the next tree. Tower of Hanoi are also similar to binary search trees since the amount of nodes in a amount of 'n' level of a tree, if you have a balanced tree is the same amount of moves it takes to solve when you have 'n' amount of disks. Lastly, we also learned that when you write a recursive program it creates a stack behind the scenes to solve all the problems, it will always go to the one on top rather than starting back at the start of the problem. If you don't have a stack you could also write it on a rewriting

machine.

2.2.2 Homework

1. Finish the Natural Number Game Addition World.

a) Show the completed work for levels 1 through 5.

b) For level 4 or 5, explain in some detail how the Lean proof is related to its corresponding proof in mathematics

1a.

Level 1:

$Sd = Sd$	<code>rfl</code>
$S(0 + d) = Sd$	<code>rw[hd]</code>
$0 + Sd = Sd$	<code>rw[add_succ]</code>
$0 = 0$	<code>rfl</code>
$0 + 0 = 0$	<code>rw[add_zero]</code>
$0 + n = n$	<code>induction n with d hd</code>

Level 2:

$SS(a + d) = SS(a + d)$	<code>rfl</code>
$S(Sa + d) = SS(a + d)$	<code>hd</code>
$S(Sa + d) = S(a + Sd)$	<code>rw[add_succ]</code>
$Sa + Sd = S(a + Sd)$	<code>rw[add_succ]</code>
$Sa = Sa$	<code>rfl</code>
$Sa = S(a + 0)$	<code>rw[add_zero]</code>
$Sa + 0 = S(a + 0)$	<code>rw[add_zero]</code>
$Sa + b = S(a + b)$	<code>induction b with d hd</code>

Level 3:

$S(d + a) = S(d + a)$	<code>rfl</code>
$S(a + d) = S(d + a)$	<code>rw[hd]</code>
$S(a + d) = Sd + a$	<code>rw[succ.add]</code>
$a + Sd = Sd + a$	<code>rw[add_succ]</code>
$a = a$	<code>rfl</code>
$a = 0 + a$	<code>rw[zero.add]</code>
$a + 0 = 0 + a$	<code>rw[add_zero]</code>
$a + b = b + a$	<code>induction b with d hd</code>

Level 4:

$S(a + (b + d)) = S(a + (b + d))$	rfl
$S(a + b + d) = S(a + (b + d))$	rw[hd]
$S(a + b + d) = a + S(b + d)$	rw[add_succ]
$S(a + b + d) = a + (b + Sd)$	rw[add_succ]
$a + b + Sd = a + (b + Sd)$	rw[add_succ]
$a + b = a + b$	rfl
$a + b = a + (b + 0)$	rw[add_zero]
$a + b + 0 = a + (b + 0)$	rw[add_zero]
$a + b + c = a + (b + c)$	induction c with d hd

Level 5:

$S(a + d + b) = S(a + d + b)$	rfl
$S(a + b + d) = S(a + d + b)$	rw[hd]
$S(a + d + b) = S(a + d) + b$	rw[succ_add]
$S(a + d + b) = a + Sd + b$	rw[add_succ]
$a + b + Sd = a + Sd + b$	rw[add_succ]
$a + b = a + b$	rfl
$a + b = a + 0 + b$	rw[add_zero]
$a + b + 0 = a + 0 + b$	rw[add_zero]
$a + b + c = a + c + b$	induction c with d hd

1b.

For Level 4, we are proving the associativity of addition. On the set of natural numbers, addition is associative. In other words, if a,b and c are arbitrary natural numbers, we have $(a+b)+c=a+(b+c)$. In Math and Lean, we have to do a proof by induction on c.

Math Proof:

$$(a + b) + c = a + (b + c)$$

Base Case: $(a+b)+0=a+(b+0)$

$(a + b) + 0 = a + (b + 0)$	
$a + b = a + (b + 0)$	def of +
$a + b = a + b$	def of +

Induction Hypothesis: $(a+b)+d=a+(b+d)$
 Induction Step: $(a+b)+Sd=a+(b+Sd)$

$$\begin{array}{ll}
 (a+b) + Sd = a + (b + Sd) & \\
 S((a+b) + d) = a + (b + Sd) & \text{def of } + \\
 S((a+b) + d) = a + S(b+d) & \text{def of } + \\
 S((a+b) + d) = S(a + (b+d)) & \text{def of } + \\
 S(a + (b+d)) = S(a + (b+d)) & \text{Induction Hypothesis}
 \end{array}$$

The Lean proof written above is the exact same as the same steps in the Math proof just backwards. Instead of having `add_zero` and `add_succ`, we have the definition of addition as that can be proven to add both successors and zero to numbers.

2.2.3 Comments and Questions

In the beginning of the reading, it takes about how recursion is different from paradox or infinite regress, since it never defines somethin in terms of itself, but always in terms of simpler versions of itself. Is the only difference between a paradox and a recursively solveable problem be that it has an exit statement at its very simplest version or are there more differences? If some paradoxs were proposed recursively, would we be able to break down some harder problems into simpler version to prove if they are unsolveable logically?

2.3 ...

...

3 Lessons from the Assignments

4 Conclusion

References

[label] Andrew Moshier, [Contemporary Discrete Mathematics](#), M&H Publishing, 2024