CPSC-354 Report

Maxwell Rovenger
Chapman University

September 8, 2024

Abstract

Contents

1 Introduction

(Delete and Replace): This report will document your learning throughout the course. It will be a collection of your notes, homework solutions, and critical reflections on the content of the course. Something in between a semester-long take home exam and your own lecture notes.[1]

To modify this template you need to modify the source `report.tex` which is available in the course repo. For guidance on how to do this read both the source and the pdf of `latex-example.tex` which is also available in the repo. Also check out the usual resources (Google, Stackoverflow, LLM, etc). It was never as easy as now to learn a new programming lanugage (which, btw, LaTeX is).

For writing LaTeX with VSCode use the LaTeX Workshop extension.

There will be deadlines during the semester, graded mostly for completeness. That means that you will get the points if you submit in time and are on the right track, independently of whether the solutions are technically correct. You will have the opportunity to revise your work for the final submission of the full report.

The full report is due at the end of the finals week. It will be graded according to the following guidelines.

[1]One purpose of giving the report the form of lecture notes is that self-explanation is a technique proven to help with learning, see Chapter 6 of Craig Barton, How I Wish I'd Taught Maths, and references therein. In fact, the report can lead you from self-explanation (which is what you do for the weekly deadline) to explaining to others (which is what you do for the final submission). Another purpose is to help those of you who want to go on to graduate school to develop some basic writing skills. A report that you could proudly add to your application to graduate school (or a job application in industry) would give you full points.

Grading guidelines (see also below):

- Is typesetting and layout professional?
- Is the technical content, in particular the homework, correct?
- Did the student find interesting references [BLA] and cites them throughout the report?
- Do the notes reflect understanding and critical thinking?
- Does the report contain material related to but going beyond what we do in class?
- Are the questions interesting?

Do not change the template (fontsize, width of margin, spacing of lines, etc) without asking your first.

2 Week by Week

2.1 Week 1

Notes

Lectures introduced Lean, a programming language to help prove discrete mathematics proofs. Through using the Natural Numbers Game, we saw how Lean operates and how it works it can, with functions acting as steps, prove theorems similar to how we did in discrete math with pen and paper and induction.

Homework

Solved problems in Natural Numbers Game using Lean to help recap teachings from Discrete Mathematics. Specifically dealing with successors and predecessors and how they can be used to change certain sides to equal the other.

Level 5/8:

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

Level 6/8:

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

Level 7/8:

```
succ n = n + 1
rw[one_eq_succ_zero]
succ n = n + succ 0
rw[add_succ]
succ n = succ (n + 0)
```

```
rw[add_zero]
succ n = succ n
rfl
```

Level 8/8:

```
2 + 2 = 4
rw[four_eq_succ_three]
2 + 2 = succ 3
rw[three_eq_succ_two]
2 + 2 = succ (succ 2)
rw[two_eq_succ_one]
succ 1 + succ 1 = succ (succ (succ 1))
rw[add_succ]
succ (succ 1 + 1) = succ (succ (succ 1))
rw[succ_eq_add_one]
succ 1 + 1 + 1 = succ (succ (succ 1))
rw[succ_eq_add_one]
1 + 1 + 1 + 1 = succ (succ (1 + 1))
rw[succ_eq_add_one]
1 + 1 + 1 + 1 = succ (1 + 1) + 1
rw[succ_eq_add_one]
1 + 1 + 1 + 1 = 1 + 1 + 1 + 1
rfl
```

For level 5/8 specifically, we can see that the Lean proof, `rw[add_zero]`, corresponds to Proof Algorithm 1: Addition, in that any variable added to zero will ultimately equal just the variable. For example: $a + 0 = a$.

From this homework, I learned how to use the Lean proof and saw how each function operated exactly like algorithms and proofs I had used in Discrete Mathematics.

Comments and Questions

Although this week generally served as just an introduction to the curriculum and a recap of discrete mathematics, I am curious as to whether or not there was a better example to be shown of how exactly a computer uses discrete mathematics. At this point, I am knowledgeable of how discrete mathematics operates and how computers can use operators to conduct mathematical operations, but I have yet to see a direct example of a computer ”thinking” through a math calculation.

If the way computers have been taught mathematics is based purely on successors and predecessors, does that make other operations like multiplication, division, and exponents far more taxing on a CPU since they have to calculate an incredible amount of successors, or do the operators used to cause multiplication and exponentiation ignore that by just creating duplicates and adding them together?

2.2 Week 2

Notes

Recursion, in coding, allows for more simplistic code that is easier to read, scale, and apply.

Regarding the allowing of typos in coding, code should be non-ambigious because coding should be universal and should work the same within the same virtual machine regardless of how you access it. We see examples of this in GitHub’s copilot since it only makes suggestions that are non-ambigious and can work on any machine, as long as they have the correct virtual machine.

Lean contains tactics, Ex: rw, and theorems, Ex: one_eq_succ_zero. Tactics are commands while theorems are logical propositions.

Homework

Level 1/5:

```
0 + n = n
induction n with d hd
0 + 0 = 0
rw[add_zero]
0 = 0
rfl
0 + succ d = succ d
rw[add_succ]
succ (0 + d) = succ d
rw[hd]
succ d = succ d
rfl
```

Level 2/5:

```
succ a + b = succ (a + b)
induction b with b
succ a + 0 = succ (a + 0)
rw[add_zero]
succ a = succ (a + 0)
rw[add_zero]
succ a = succ a
rfl
succ a + succ b = succ (a + succ b)
rw[add_succ]
succ (succ a + b) = succ (a + succ b)
rw[n_ih]
succ (succ (a + b)) = succ (a + succ b)
rw[add_succ]
succ (succ (a + b)) = succ (succ (a + b))
rfl
```

Level 3/5:

```
a + b = b + a
induction b with b hb
a + 0 = 0 + a
rw [add_zero]
a = 0 + a
rw [zero_add]
a = a
rfl
a + succ b = succ b + a
rw [add_succ]
succ (a + b) = succ b + a
rw [hb]
succ (b + a) = succ b + a
rw [succ_add]
```

```
succ (b + a) = succ (b + a)
rfl
```

Level 4/5:

```
a + b + c = a + (b + c)
induction c with c hc
a + b + 0 = a + (b + 0)
rw [add_zero]
a + b = a + (b + 0)
rw [add_zero]
a + b = a + b
rfl
a + b + succ c = a + (b + succ c)
rw [add_succ]
succ (a + b + c) = a + (b + succ c)
rw [add_succ]
succ (a + b + c) = a + succ (b + c)
rw [add_succ]
succ (a + b + c) = succ (a + (b + c))
rw [hc]
succ (a + (b + c)) = succ (a + (b + c))
rfl
```

Level 5/5:

```
a + b + c = a + c + b
induction c with c hc
a + b + 0 = a + 0 + b
rw [add_zero, add_zero]
a + b = a + b
rfl
a + b + succ c = a + succ c + b
rw [add_succ]
succ (a + b + c) = a + succ c + b
rw [add_succ]
succ (a + b + c) = succ (a + c) + b
rw [succ_add]
succ (a + b + c) = succ (a + c + b)
rw [hc]
succ (a + c + b) = succ (a + c + b)
rfl
```

Comments and Questions

Personally, I felt that when it comes to AI autocorrecting code, instead of being lenient towards mistyped code, AI could just simply correct the typos. This would allow for a smoother coding process wihle also creating non-ambigious code.

After playing around with the Tower of Hanoi, it is apparent that recursion is invaluable in programming. We often see it as a way to neatly write code but it can also be massive time saver as apparent with the Tower of Hanoi where instead of having to, through trial and error finding out the solution, we can just use a recursive equation and plug in our values. My question then is, in what other fields can we apply recursion, as it is becoming more apparent to me that recursion can be a form of thinking rather than a field specific method?

Lean Proof

$$\begin{aligned} a+b+c &= a+(b+c) \\ a+b+0 = a+(b+0) &== def of + \\ a+b = a+(b+0) &== def of + \\ a+b &= a+b \end{aligned}$$

2.3 Week 3

Notes

Homework

Comments and Questions

...

3 Lessons from the Assignments

(Delete and Replace): Write three pages about your individual contributions to the project.

On 3 pages you describe lessons you learned from the project. Be as technical and detailed as possible. Particularly valuable are *interesting* examples where you connect concrete technical details with *interesting* general observations or where the theory discussed in the lectures helped with the design or implementation of the project.

Write this section during the semester. This is approximately a quarter of apage per week and the material should come from the work you do anyway. Just keep your eyes open for interesting lessons.

Make sure that you use LaTeX to structure your writing (eg by using subsections).

4 Conclusion

(Delete and Replace): (approx 400 words) 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 software engineering? What did you find most interesting or useful? What improvements would you suggest?

References

[BLA] Author, Title, Publisher, Year.