CPSC-354 Report

John Robert Mulhern
Chapman University

September 29, 2024

Abstract

This document will contain the various assignments completed by John Mulhern over the course of the CPSC 354 Programming Languages course. For any questions, comments, or concerns in this document, feel free to reach out to John Mulhern at his email, mulhern@chapman.edu, or by phone number: (208)-451-3484.

Contents

1 Introduction

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

To modify this template I would 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.

[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.

There will be deadlines during the semester, graded mostly for completeness. That means that I will get the points if I submit in time and are on the right track, independently of whether the solutions are technically correct. I will have the opportunity to revise my 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.

Grading guidelines (see also below):

- Is typesetting and layout professional?
- Is the technical content, in particular the homework, correct?
- Did I 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

Tuesday: Orientation and introduction to the course.
Thursday: First Lab on Tuesday's content.

Notes and Homework

Our homework for this week was to finish levels 5-8 of the tutorial world inside the Natural Numbers Game provided to us in class. On Tuesday of week 1, we went over the game in basic detail, covering levels 1-4 as to become used to the website so we could begin our first challenge. The solutions to each of the worlds can be found below in an itemized format.

- `World 5 Solution: rw[add_zero], rw[add_zero], rfl.`
- `World 6 Solution: rw[add_zero c], rw[add_zero b], rfl.`
- `World 7 Solution: rw[one_eq_succ_zero], rw[add_succ], rw[add_zero], rfl.`
- `World 8 Solution: rw[two_eq_succ_one, one_eq_succ_zero], rw[add_succ], rw[add_zero], rw[<- one_eq_succ_zero, <- two_eq_succ_one, <- three_eq_succ_two, <- four_eq_succ_three]`

World 5's solutition is to rewrite the equation by adding zero onto a variable **b** or **c**, and as we know from a descrete math proof described as $\forall n \in N, n+0 = n$, adding zero to any number provides the same equivalent number. This becomes useful later in sections seven and eight when we begin dealing with adding zero to successor values.

Comments and Questions

Looking at Discrete Math over the past few days and getting a refresher on the course since I took it a few semesters ago has been very interesting. I have enjoyed the tutorial levels of the Natural Numbers Game, and I particularly enjoyed their explanations for the proofs and early concepts for Discrete Math. Had I known about this website when I was taking the course, I have a feeling it would have been a great resource to support my understanding of those proofs and other concepts.

My question then in relation to discrete mathematics comes more so with how we utilize those proofs on paper verses when they are used in a computational environment. For example, writing a Discrete Math proof can often take a significant amount of time and paper to create, depending of course on the operation. For something as simple as addition or multiplication, using the proofs outlined in Discrete Math can turn a simple problem, such as $2 * (3 + 2 + 4)$, into a massive multi-page proof. However, when a computer runs such a problem, it concludes the correct answer in an astoundingly short amount of time.

My question is then, what is the largest, hardest, and most difficult proof a person could do by hand that can be done in seconds by a machine?[2]

2.2 Week 2

Tuesday: What is a proposition? Covering Eric Villanueva's question "I wonder how the computer or code implements the logic we have in understanding discrete mathematics to make computtations. How do we define the idea of successors so that the computer knows how to carry out calculations?"

Notes and Homework

- `World 1 Solution: Induction n with d, rw[add_zero], rfl, rw[succ_eq_add_one], rw[one_eq_succ_zero] rw[add_zero], rw[add_succ], rw[n_ih], rfl.`
- `World 2 Solution: Induction b with d hd, rw[add_zero, add_zero], rfl, rw[add_succ, add_succ], rw[hd], rfl.`
- `World 3 Solution: Induction b with b hb, rw[add_zero, zero_add], rfl, rw[add_succ, succ_add, hb], rfl.`
- `World 4 Solution: Induction c with c hc, rw[add_zero], rw[add_zero], rfl, rw[add_succ], rw[add_succ], rw[hc], rfl.`
- `World 5 Solution: Induction c with c hc, rw[add_zero, add_zero], rfl, rw[add_succ, add_succ], rw[succ_add], rw[hc], rfl.`

World 5's solution is to discover the add_right_comm function through other theorems we are already familier with. Utilizing several Lean theorems, such as `rw[add_succ], rw[add_zero]`, and proving by indcution, we can rewrite $a + b + c = a + c + b$ into $\text{succ}(a + c + b) = \text{succ}(a + c + b)$. Proving this fact using standard mathematics follows a similar set of rules as well. Should you aim to prove by induction, you can then add zeros and use reflexivity to reconstruct an equation to prove add_right_comm.

Comments & Questions

In reference to the additional reading for this week, the dialogue *Little Harmonic Labrinth*, it describes recursion in a much more palatable format. Essentially, you have an item operating within itself endlessly. This concept is very useful as a time saver when it comes to certain coding tasks, but my question in relation to recursion is; How can recursion be optimized in a computer environment as to be effective without running out of resources in relation to large tasks?

[2]It is important to learn to ask *interesting* questions. There is no precise way of defining what is meant by interesting. You can only learn this by doing. An interesting question comes typically in two parts. Part 1 (one or two sentences) sets the scene. Part 2 (one or two sentences) asks the question. A good question strikes the right balance between being specific and technical on the one hand and open ended on the other hand. A question that can be answered with yes/no is not an interesing question.

2.3 Week 3

Notes & Homework

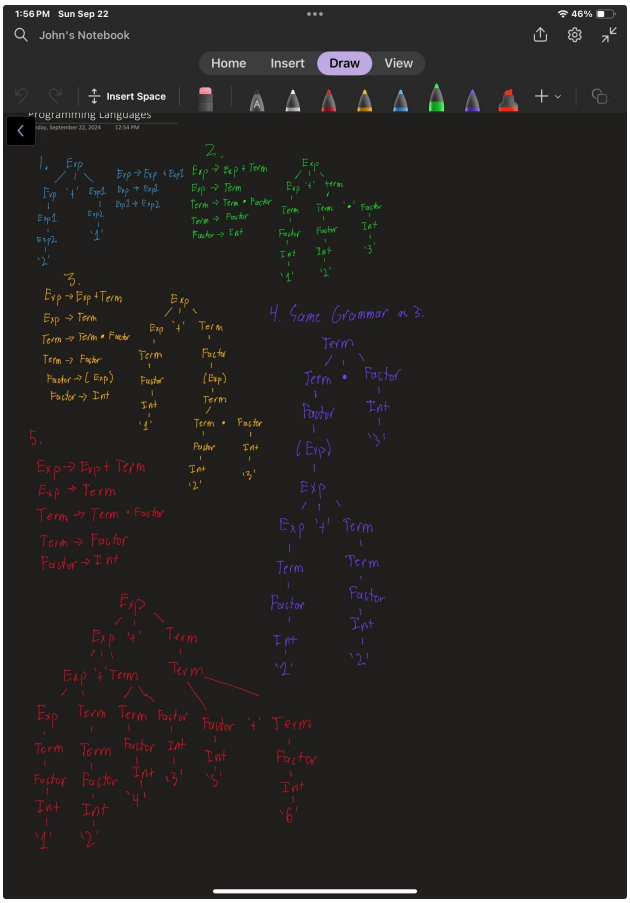
Comments & Questions

2.4 Week 4

Notes & Homework

Tuesday: Covering individuals projects from last week. Thursday: Learning to use Cursor and its various features. Progress on Assignment 1 and associated lab work.

Homework: Solve homework exercises in relation to parsing trees. Below, questions 1-5 are completed with specific solutions being in differing colors.



Comments & Questions

My question for this week centers around the ideas of parsing trees. How are parsing trees best utilized when in a coding environment? They seem very useful for mathematical equations and logical reasoning, but I am unsure of how they would be useful when programming.

2.5 Week 5

Notes & Homework

Tuesday: Working on Assignment 2, due Wednesday. Thursday: Begin ∧ Tutorial: Party Invites. Solutions to levels 1-8 are listed below.

- `World 1 Solution:` exact $todo\_list$.
- `World 2 Solution:` exact$\langle p, s\rangle$.
- `World 3 Solution:` have $ai := and\_introai, haveou := and\_introou, exact\langle ai, ou\rangle$.
- `World 4 Solution:` exact and_left vm.
- `World 5 Solution:` exact and_right h.
- `World 6 Solution:` exact$\langle h1.left, h2.right\rangle$.
- `World 7 Solution:` have $h1 := h.left$, have $h2 := h.right$, have $h3 := h2.left$, have $h4 := h3.left$, exact $h4.right$.
- `World 8 Solution:` have $h1 := h.left$, have $a := h1.right$, have $b := h1.left$, have $h2 := h.right$, have $h3 := h2.right$, have $h4 := h3.left$, have $c := h4.left$, exact$\langle a, b, c\rangle$.

The solution for world 8 written in mathematical logic goes as follows:

- Assume $h = ((P \wedge S) \wedge A) \wedge \not{I} \wedge (C \wedge \not{O}) \wedge \not{U}$
- $h1 = (P \wedge S) \wedge A$ and_left on h (1)
- $a = A$ and_right on h1 (2)
- $b = P \wedge S$, and_left on h1 (3)
- $h2 = \not{I} \wedge (C \wedge \not{O}) \wedge \not{U}$ and_right on h (4)
- $h3 = (C \wedge \not{O}) \wedge \not{U}$, and_right on h2 (5)
- $h4 = C \wedge \not{O}$, and_left on h3 (6)
- $c = C$, and_left on h4 (7)
- $A \wedge C \wedge P \wedge S$ (8)

Comments & Questions

My question for this week pertains to the complexity of lean. As we progress through different worlds of the game, the harder and more complex the problems we have to solve, especially on boss levels, where combinations from prior levels are brought together to make one much more challenging level. In math however, items that are significantly more complex than simple logic proofs or basic arithmetic exist. How then does Lean simplify complex math down into assumed proofs for much larger and complex equations or logic proofs, and what problems take longer to solve using a Lean logic rather than standard math or coding logic?

2.6 ...

...

3 Lessons from the Assignments

4 Conclusion

References

[BLA] Author, Title, Publisher, Year.