CPSC-402 Report
Compiler Construction

Connor Cowher
Chapman University

May 16, 2022

Abstract

Short summary of purpose and content.

Contents

1 Introduction **1**
1.1 General Remarks . . . 1
1.2 LaTeX Resources . . . 2
1.2.1 Subsubsections . . . 2
1.2.2 Itemize and enumerate . . . 2
1.2.3 Typesetting Code . . . 2
1.2.4 Including Graphics . . . 3
1.2.5 More Mathematics . . . 3
1.2.6 Definitons, Examples, Theorems, Etc . . . 3
1.3 Plagiarism . . . 3

2 Homework **3**
2.1 Week 1 . . . 4
2.2 Week 2 . . . 5
2.3 Week 3 . . . 6
2.4 Week4 . . . 8
2.5 Week10 . . . 10

3 Project **11**

4 Conclusions **11**

1 Introduction

Replace this entire Section 1 with your own short introduction.

1.1 General Remarks

First you need to download and install LaTeX.[1] For quick experimentation, you can use an online editor such as Overleaf. But to grade the report I will used the time-stamped pdf-files in your git repository.

[1] Links are typeset in blue, but you can change the layout and color of the links if you locate the `hypersetup` command.

LaTeX is a markup language (as is, for example, HTML). The source code is in a `.tex` file and needs to be compiled for viewing, usually to `.pdf`.

If you want to change the default layout, you need to type commands. For example, `\medskip` inserts a medium vertical space and `\noindent` starts a paragraph without indentation.

Mathematics is typeset between double dollars, for example

$$x + y = y + x.$$

1.2 LaTeX Resources

I start a new subsection, so that you can see how it appears in the table of contents.

1.2.1 Subsubsections

Sometimes it is good to have subsubsections.

1.2.2 Itemize and enumerate

- This is how you itemize in LaTeX.
- I think a good way to learn LaTeX is by starting from this template file and build it up step by step. Often stackoverflow will answer your questions. But here are a few resources:
  1. Learn LaTeX in 30 minutes
  2. LaTeX – A document preparation system

1.2.3 Typesetting Code

A typical project will involve code. For the example below I took the LaTeX code from stackoverflow and the Haskell code from my tutorial.

```
-- run the transition function on a word and a state
run :: (State -> Char -> State) -> State -> [Char] -> State
run delta q [] = q
run delta q (c:cs) = run delta (delta q c) cs
```

Short snippets such as `run :: (State -> Char -> State) -> State -> [Char] -> State` can also be directly fitted into text. There are several ways of doing this, for example, `run :: (State -> Char -> State) -> State ->` is slightly different in terms of spaces and linebreaking (and can lead to layout that is better avoided), as is

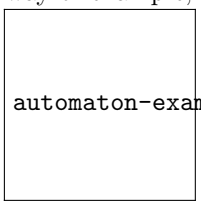
`run :: (State -> Char -> State) -> State -> [Char] -> State`

For more on the topic see Code-Presentations Example.

Generally speaking, the methods for displaying code discussed above work well only for short listings of code. For entire programs, it is better to have external links to, for example, Github or Replit (click on the "Run" button and/or the "Code" tab).

1.2.4 Including Graphics

By way of example, I include here a screenshot from the book [HMU].



You can use the same technique to include any handwritten drawings.

1.2.5 More Mathematics

We have already seen $x + y = y + x$ as an example of inline maths. We can also typeset mathematics in display mode, for example

$$\frac{x}{y} = \frac{xy}{y^2},$$

Here is an example of equational reasoning that spans several lines:

$$\begin{aligned} \text{fib}(3) &= \text{fib}(1) + \text{fib}(2) && \text{fib}(n+2) = \text{fib}(n) + \text{fib}(n+1) \\ &= \text{fib}(1) + \text{fib}(0) + \text{fib}(1) && \text{fib}(n+2) = \text{fib}(n) + \text{fib}(n+1) \\ &= 1 + 0 + 1 && \text{fib}(0) = 0, \text{fib}(1) = 1 \\ &= 2 && \text{arithmetic} \end{aligned}$$

1.2.6 Definitons, Examples, Theorems, Etc

Definition 1.1. This is a definition.

Example 1.2. This is an example.

Proposition 1.3. *This is a proposition.*

Theorem 1.4. *This is a theorem.*

You can also create your own environment, eg if you want to have Question, Notation, Conjecture, etc.

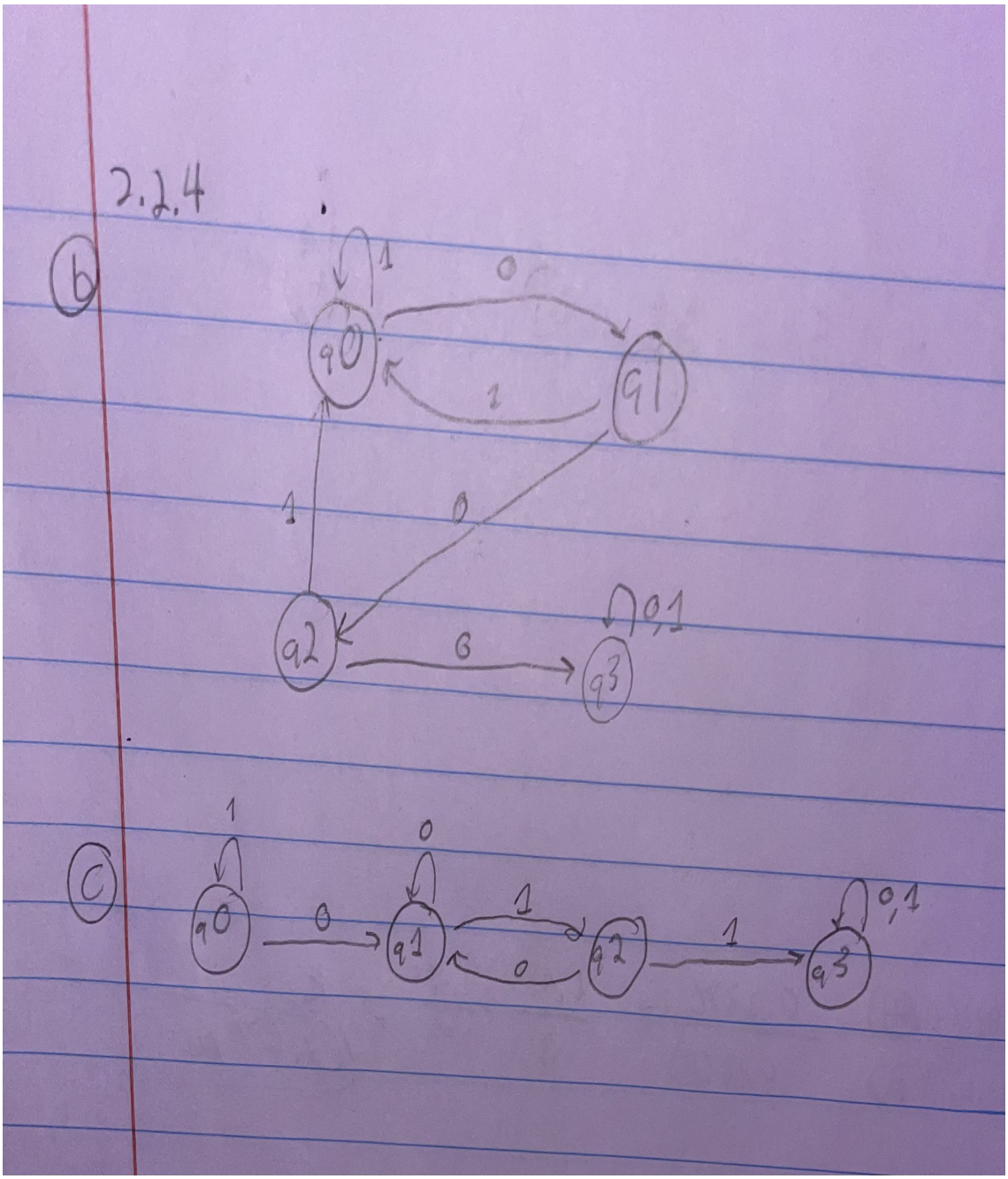
1.3 Plagiarism

To avoid plagiarism, make sure that, in addition to my course notes, you also cite all the external sources you use. Make sure you cite your references throughout your text, not only at the end.

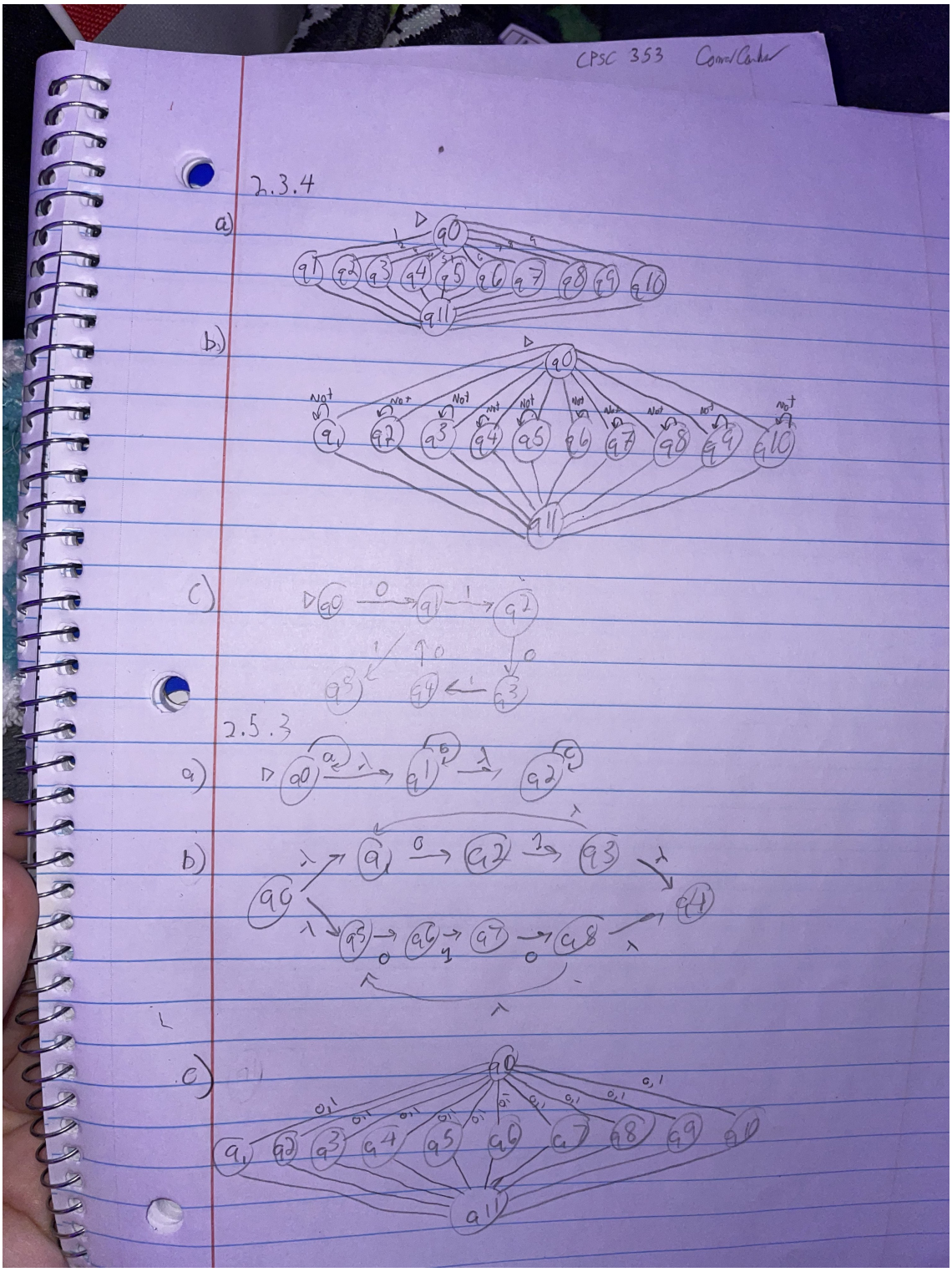
2 Homework

This section will contain your solutions to homework.

2.1 Week 1

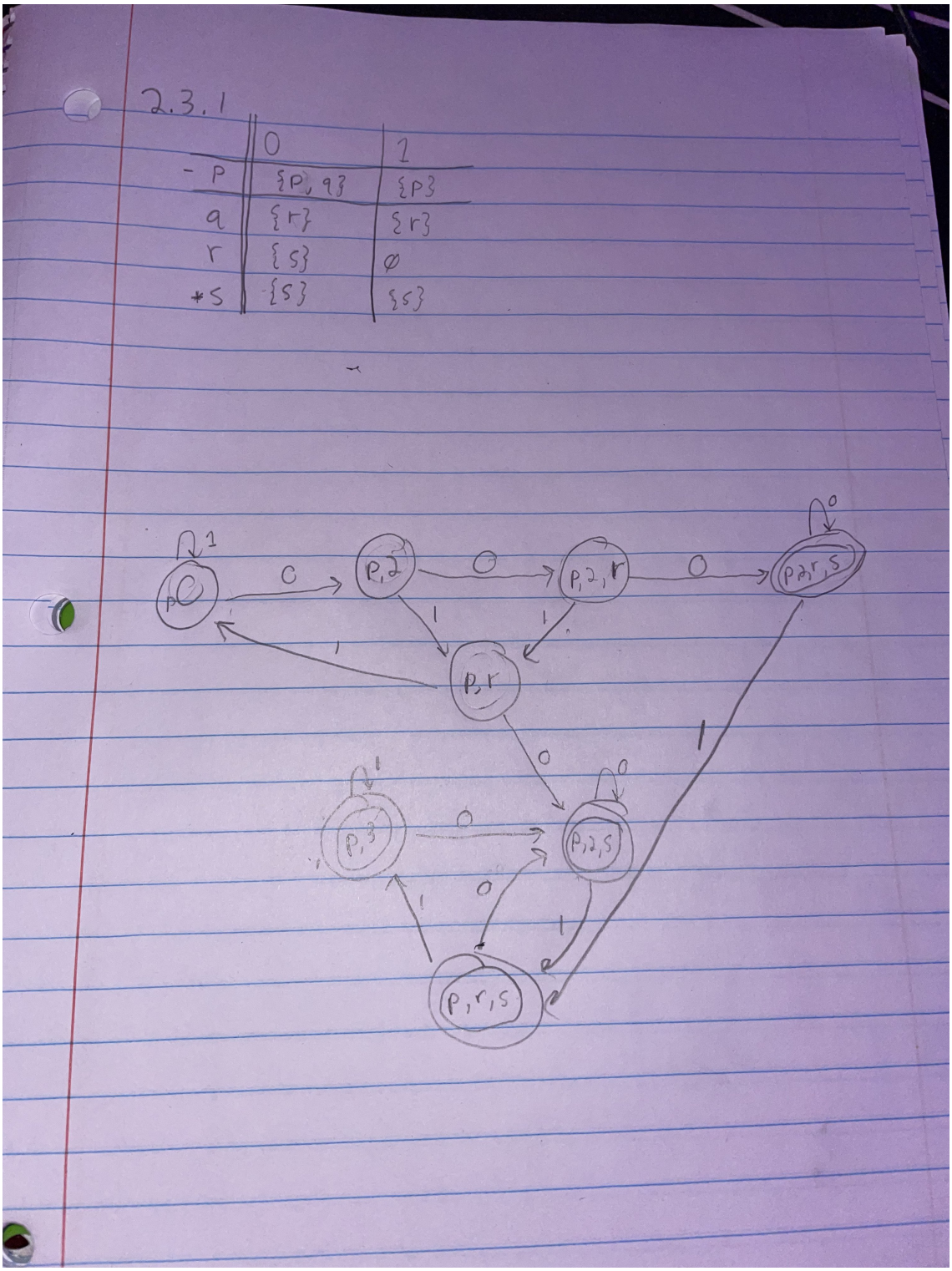


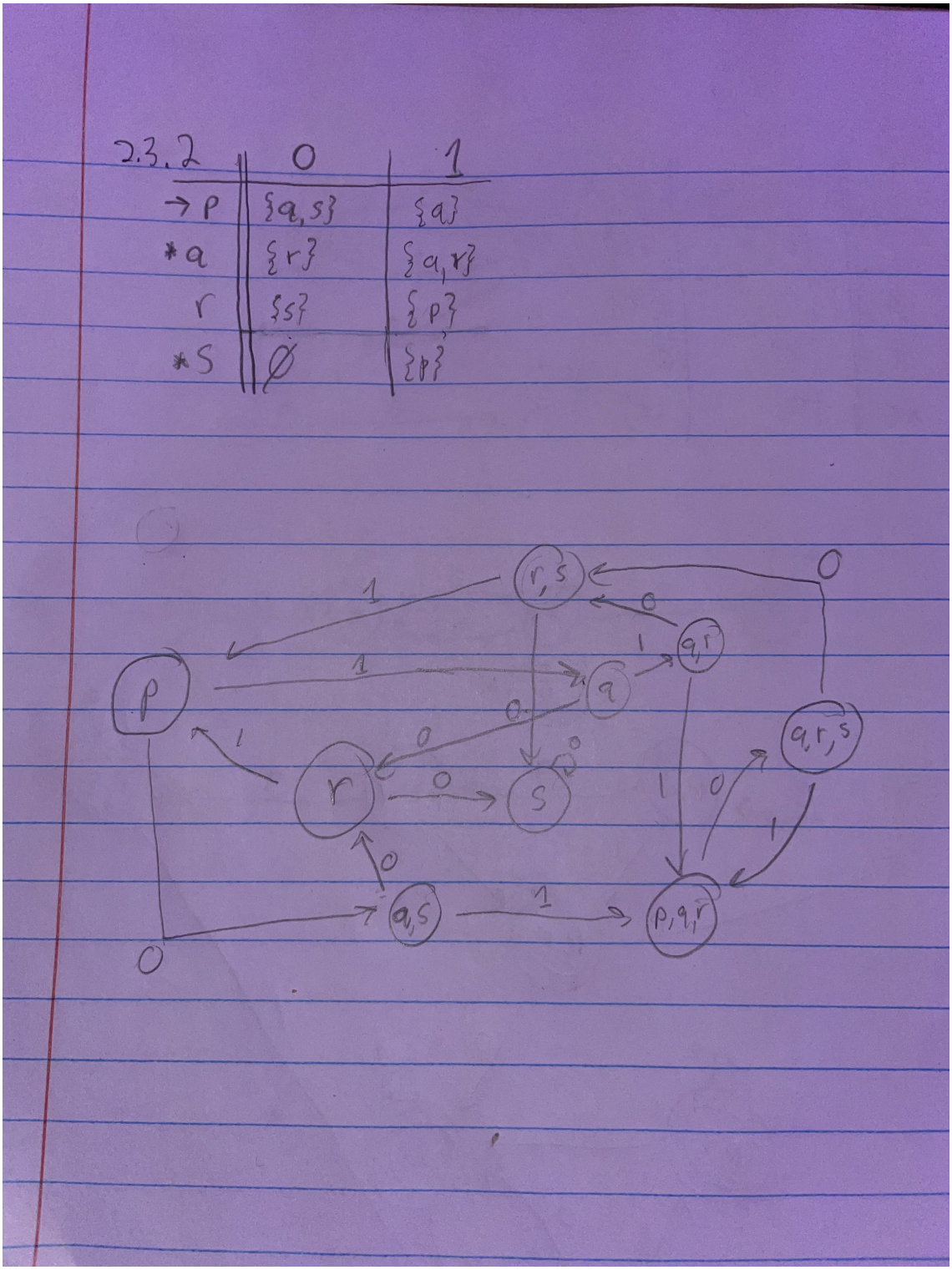
2.2 Week 2



2.3 Week 3

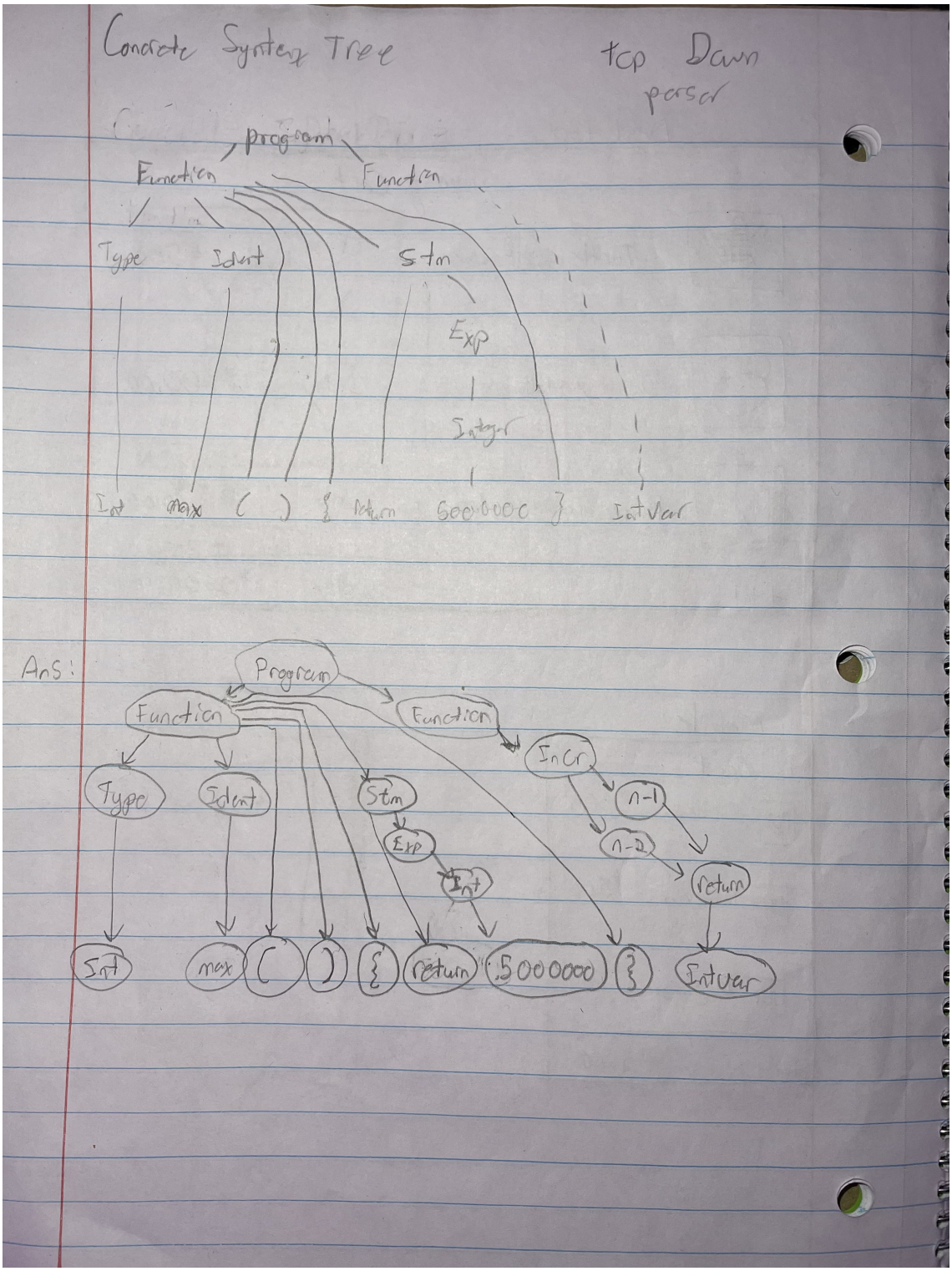
Converting NFAs to DFAs by Hand



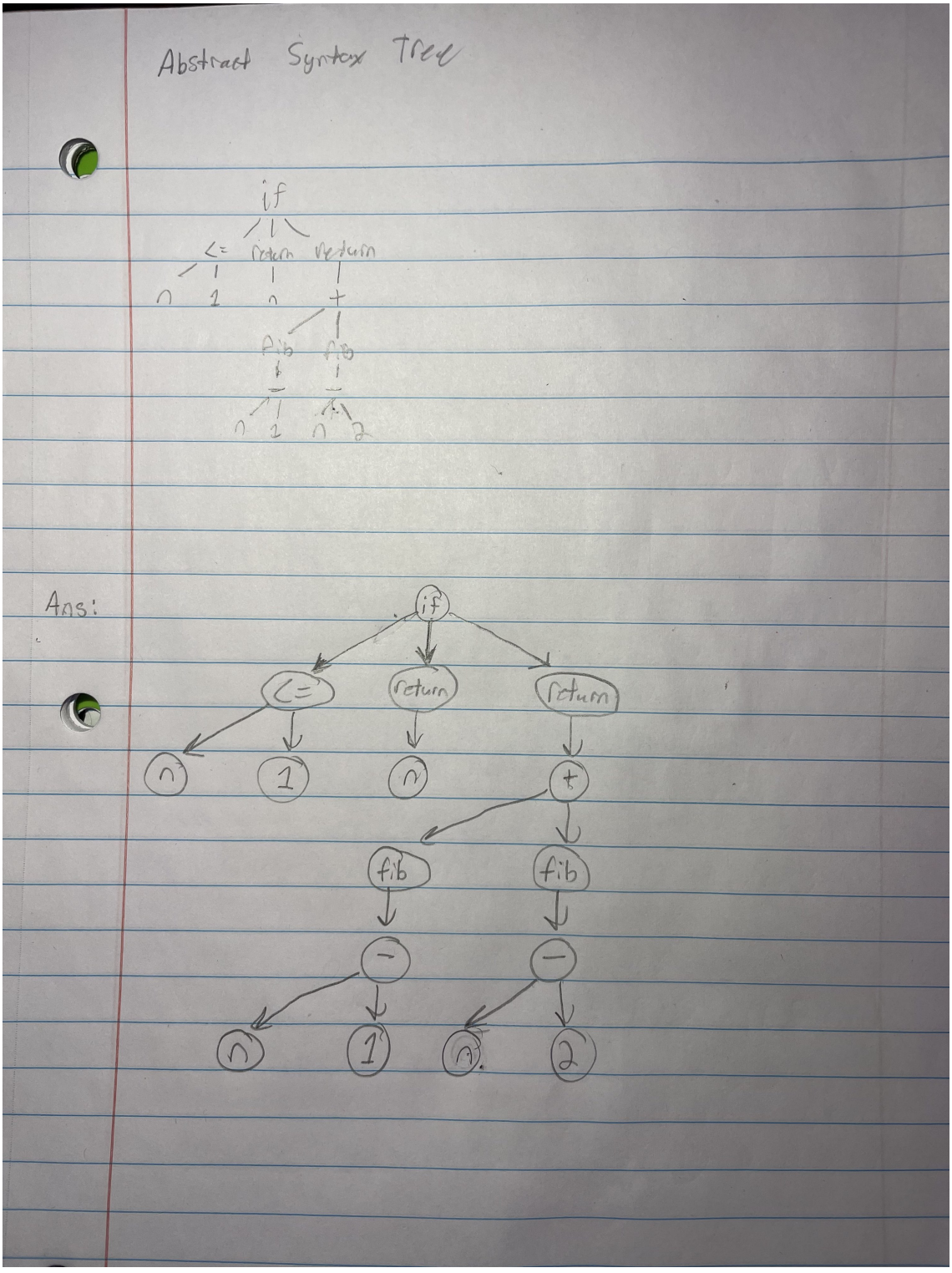

2.3.2
0
1
→p
{q,s}
{q}
*q
{r}
{q,r}
r
{s}
{p}
*s
∅
{p}
p
q
r
s
r,s
q,r
q,r,s
q,s
p,q,r
0
1

2.4 Week4

Write out the concrete syntax tree for the complete Fibonacci program.

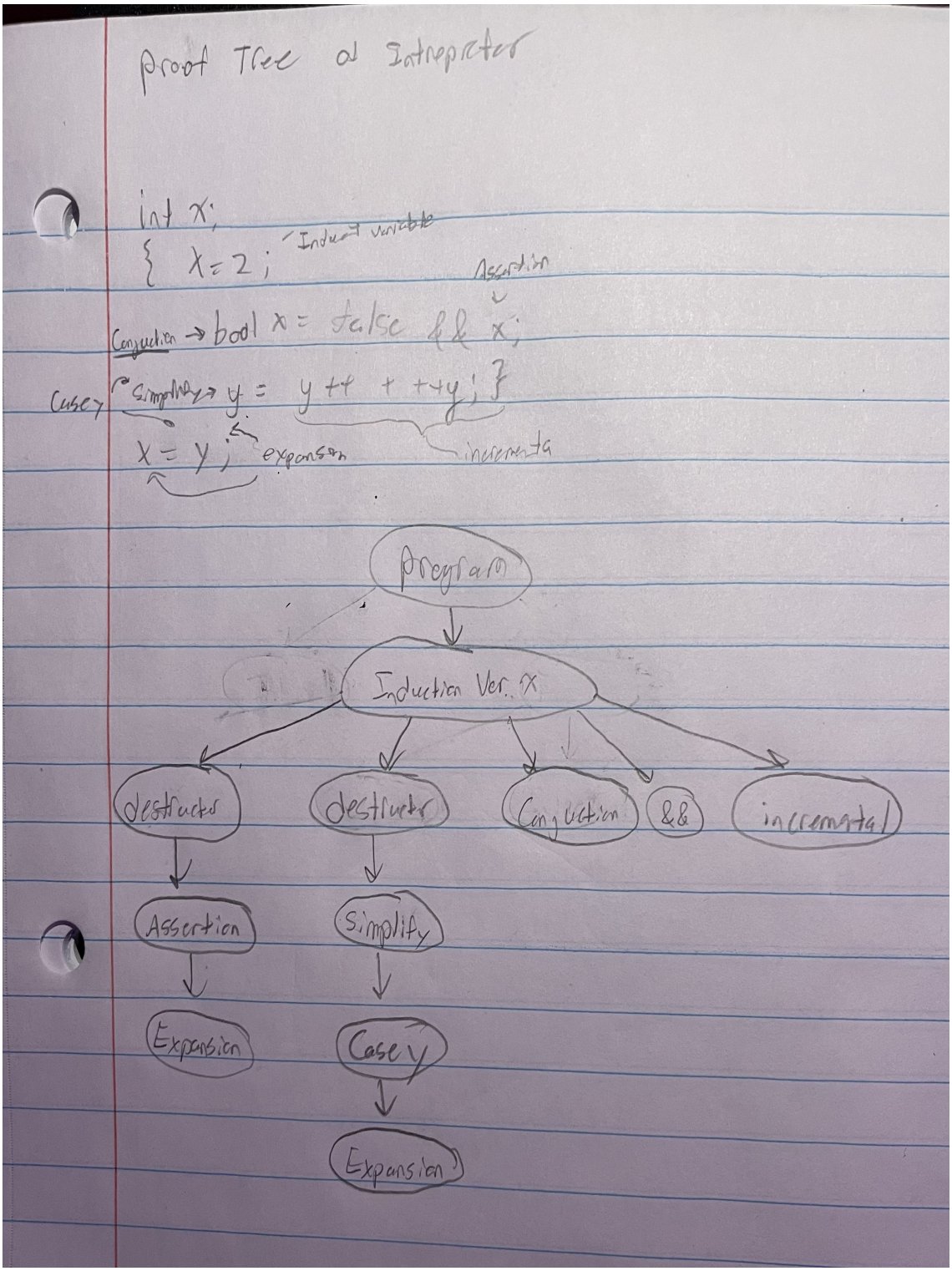


Write out the abstract syntax tree for the complete Fibonacci program.



2.5 Week10

Show, in the form of a proof tree, the steps taken by an interpreter evaluating the following program fragment.



3 Project

The project will be to compile code from a programming language of your choice to assembly and to explain the assembly code and compilation process using the knowledge your learned in this course.

Pro tips:

- Choose a compiled (not interpreted) programming language.
- Choose a language that you find interesting anyway.
- Start early.
- Come to office hours. I have not run this part of the course before and I am really interested what you will find.

4 Conclusions

(approx 400 words)

In the conclusion, I want 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 programming languages and software engineering?

References

[HMU] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman: Introduction to automata theory, languages, and computation, 3rd Edition. Pearson international edition, Addison-Wesley 2007