CPSC-406 Report

Erik Van Cruyningen
Chapman University

April 24, 2025

Abstract

Contents

1 Introduction

2 Week by Week

2.1 Week 1

2.1.1 Homework 1: Introduction to Automata

Problem 1: Characterize All Words to Sum 25 Cents

All words that end in the accepting state follow a pattern. They can be one of three things: Five fives, three fives and one ten, or one five and two tens. This can be written as:

$$(5^5)|(5^310)|(5^110^2)$$

This is every possible combination of valid answers, but answers over 25 cents are rejected.

Problem 2: Create a Regular Expression to express the pattern in the image

The pattern consists of any number of pushes and pays followed by at least one pay. Following the guidelines in the homework, the regex is:

$$(push + pay)^*pay$$

2.1.2 Homework 1: DFAs and NFAs

Problem 1: Determine whether the following words belong to L1, L2, or L3.

$$L1 := \{x01y \mid x, y \in \Sigma^*\}$$
$$L2 := \{w \mid |w| = 2^n, n \in \mathbb{N}\}$$
$$L3 := \{w \mid |w_0| = |w_1|\}$$

L1 accepts any word that contains "01" in it. L2 accepts words where the length is a power of 2. L3 accepts any word where the word is of equivalent length to the word w0.

	L1	L2	L3
w1 = 10011	1	0	1
w2 = 100	0	0	0
w3 = 10100100	1	1	0
w4 = 1010011100	1	0	0
w5 = 11110000	0	1	0

Problem 2: Determining Accepted States

We have a DFA defined as follows:

1. $Q = \{q_0, q_1, q_2\}$
2. $\Sigma = \{0, 1\}$
3. Transition function:

$$\delta(q_0, 0) = q_2, \qquad \delta(q_0, 1) = q_0,$$
$$\delta(q_2, 0) = q_2, \qquad \delta(q_1, 1) = q_0,$$
$$\delta(q_1, 0) = \delta(q_1, 1) = q_1$$

4. Initial state: q_0

5. Accepting state: $F = \{q_1\}$

For the words $w_1 = 0010$, $w_2 = 1101$, and $w_3 = 1100$, only w_1 and w_2 end in the accepted state q_1.

2.1.3 Exploration

I believe this material is included in the course because sets up the formal definitions of DFAs and NFAs that we can build off of later in the course. This material is quite interesting to me, because it seems you can build an automata for many things we use in our everyday life. My coffee maker has a selection of inputs, a beginning state, and a goal state, and I think it follows the definition of a DFA. Defining everyday objects as automata makes me understand coursework and comprehend the course better, as I can apply what we learn to everyday objects.

2.1.4 Questions and Comments

How common are objects in our lives that follow a DFA scheme? For example, an app on a phone could be seen as automata, and perhaps Instagram's "Goal State" would be keeping the app open for as long as possible.

2.2 Week 2

2.2.1 Homework 2: Automata

Problem 1: Word Processing with DFAs

Below is a table of example words and whether they are accepted by A1 and A2 as shown inside the homework.

w	A1	A2
w1 = AAA	0	1
w2 = AAB	1	0
w3 = ABA	0	0
w4 = ABB	0	0
w5 = BAA	0	1
w6 = BAB	0	0
w7 = BBA	0	0
w8 = BBB	0	0

Describe the language L(A(k)) accepted by A(k) for both k=1 and k=2:
A1 needs to start with an a, and then have an odd number of b's and end in a b.
A2 needs to end in at least two consecutive a's.

Problem 2: Implementing DFA runs

We begin by implementing the run method, which should begin by setting the current state to the initial state of the DFA. It then should go through all the characters in the passed in word, and update its state depending on the input and transition function. When it has finished running, it checks whether the current state is inside the list of final states and returns a boolean if it is in the list of final states.

```
def run(self, w):
      state = self.q0
      for char in w:
          state = self.delta[(state, char)]
      if state in self.F:
          return True
      return False
```

We then move on to defining the DFAs, which can be easily defined as a set of states, a list of accepted letters, a transition function outlining what to do with each input in each state, an initial state, and finally a set of accepting states. An example is:

```
A1 = dfa.DFA(
      Q={1, 2, 3},
      Sigma={"a", "b"},
      delta={
          (1, "a"): 2,
          (1, "b"): 4,
          (2, "a"): 2,
          (2, "b"): 3,
          (3, "a"): 2,
          (3, "b"): 2,
          (4, "a"): 4,
          (4, "b"): 4,
      },
      q0=1,
      F={3},
  )
```

Problem 3: (Designing DFAs

We implemented the DFAs into our Python program and examined what happened when we ran them with the list of all possible three letter combinations of a's and b's. All of them appeared to be correct and when we were implementing them, we saw that a lot of them shared a similar core structure and not much needed to be changed. For example, the DFA for "an odd number of a's and an odd number of b's" was identical to the DFA for "an even number of a's and an odd number of b's", except that the goal state was moved by one unit. This would apply to any even or odd counter. Another similarity was the words that contain a string of three characters, as they are identical except for the inputs.

Problem 4: (A new automata from an old one)

To make a DFA that accepts only the words that a different DFA rejects and rejects the one that the same DFA accepts, you simply need to take the original DFA and swap the accepting states and non accepting states. For example, if A0 only accepted state 3, then A1 could accept state 1 and 2 and it would be the exact opposite. The code for this follows this premise and finds all the states that are not in the original list of accepting states.

```
def refuse(self):
      N = set()
      for state in self.Q:
          if state not in self.F:
              N.add(state)

      # Create a new DFA that uses N
      newDFA = DFA(Q=self.Q, Sigma=self.Sigma, delta=self.delta, q0=self.q0, F=N)

      return newDFA
```

2.2.2 Exploration

I think this material is included in the course because it requires the students to critically think about the formal definitions of DFAs that we can use with either pure math or graphical depictions. I think it is quite cool how DFAs can be implemented so simplistically in code, and how practical they can be.

2.2.3 Questions and Comments

Is there a purpose to have the inverse of all DFAs? I understand many DFAs and their inverses have applications, but some DFAs must have an application and their inverse is completely useless

2.3 Week 3

2.3.1 Homework 3: Operations On Automata

Problem 1: Extended Transition Functions

The langauge accepted by A2 are words that are an odd number of characters in length and do not have any sequential b's.

If we look at our function, S(state, letter), we can recursively find what the state is for any given word.

$$S(1, abaa) = S(S(1, a), baa)$$

$$S(1, a) = 2$$

We then substitute to simplify:

$$S(2, baa) = S(S(2, b), aa)$$

$$S(2, b) = 4$$

Substitute again:

$$S(4, aa) = S(S(4, a), a)$$

$$S(4, a) = 2$$

Final State:

$$S(2, a) = 3$$

We then repeat the process for S2.

$$S(1, abba) = S(S(1, a), bba)$$

$$S(1, a) = 2$$

We then substitute to simplify:

$$S(2, bba) = S(S(2, b), ba)$$

$$S(2, b) = 1$$

Substitute again:

$$S(1, ba) = S(S(1, b), a)$$

$$S(1, b) = 3$$

Final State:

$$S(3, a) = 3$$

We are sure that our new DFA, L3 is the combination of L1 and L2 because if captures every possible state of the two that is reachable. Every point in L3 is a set of the two states in L1 and L2, and it captures all possible routes that can make it to that place.

In order to change L3 such that is becomes the union of L1 and L2, you would need to make a new DFA that accepts all the words that both L1 and L2 accept.

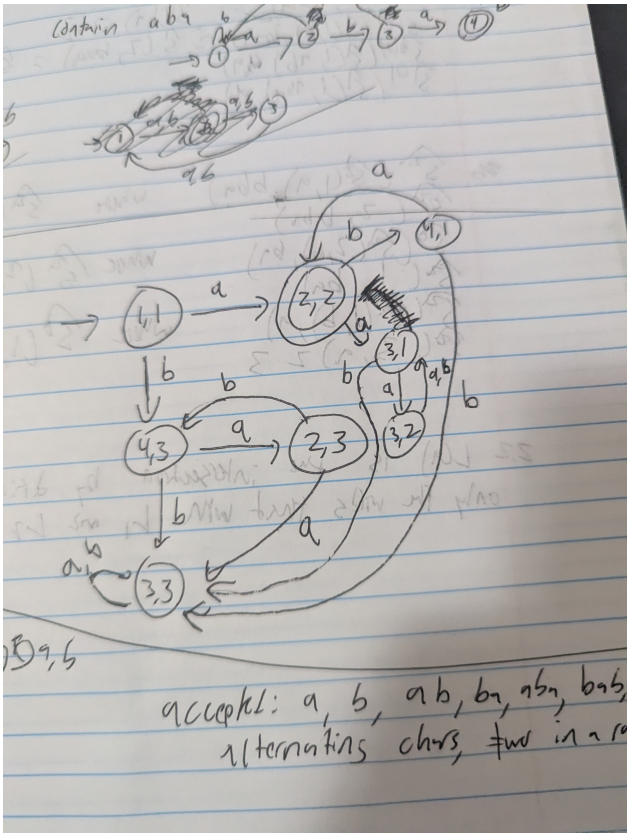

Figure 1: The $L(\mathcal{A}) = L(\mathcal{A}^{(1)}) \cap L(\mathcal{A}^{(1)})$

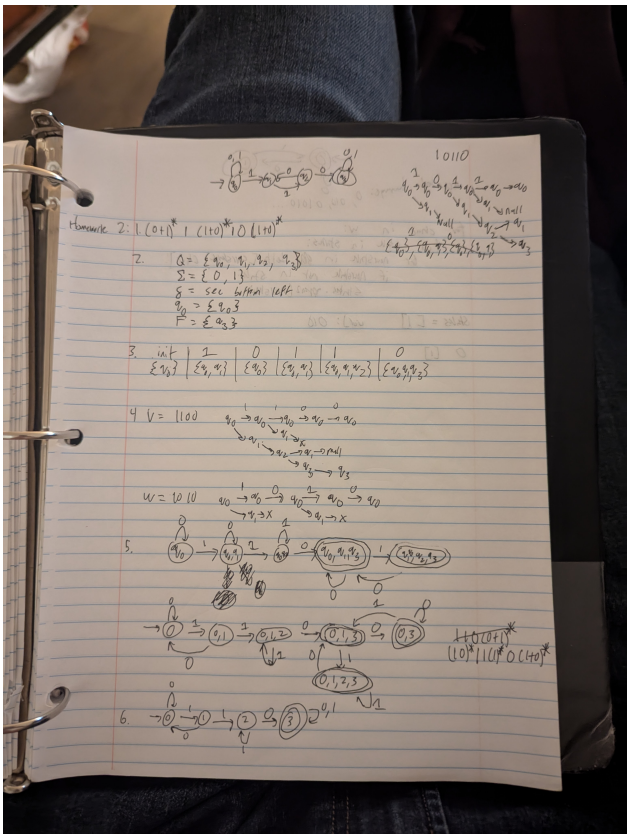

Figure 2: Homework 4, Problem 2

2.4 Week 4

2.4.1 Homework 4: Determinization

Problem 1: Explain how a DFA can be seen as an NFA

A DFA can be seen as an NFA because the transition function can be changed from a mapping of a state and an input to be an input and a set of states that it leads to. A DFA will have singleton states and the remaining features of the DFA stay the same in the new NFA. Because every DFA is an NFA by definition, L(A) will equal L(A') every time.

Problem 2: Describing NFAs. See Fig 2.

2.4.2 Exploration

I like this material in the course, because I can see how an NFA would be more useful than a DFA, but harder to implement in code. I am working on the first project, and it is quite stimulating to think about the changes required to alter an NFA to become a DFA.

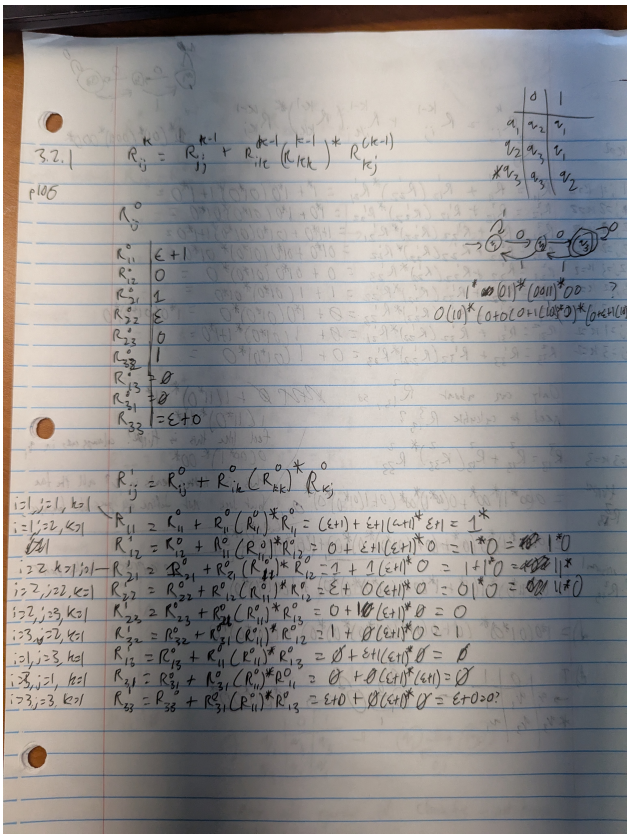

Figure 3: 3.2.1 Part 1

2.4.3 Questions and Comments

Are there some things that regular expressions cannot solve? It is really good at pattern matching and parsing text, but I cannot imagine that it would be good at natural language text.

2.5 Week 4

2.5.1 Homework 5: ITALC Problems

Problem 1: ITALC 3.2.1

Problem 2: ITALC 3.2.2

Problem 3: ITALC 4.4.1

Problem 4: ITALC 4.4.2

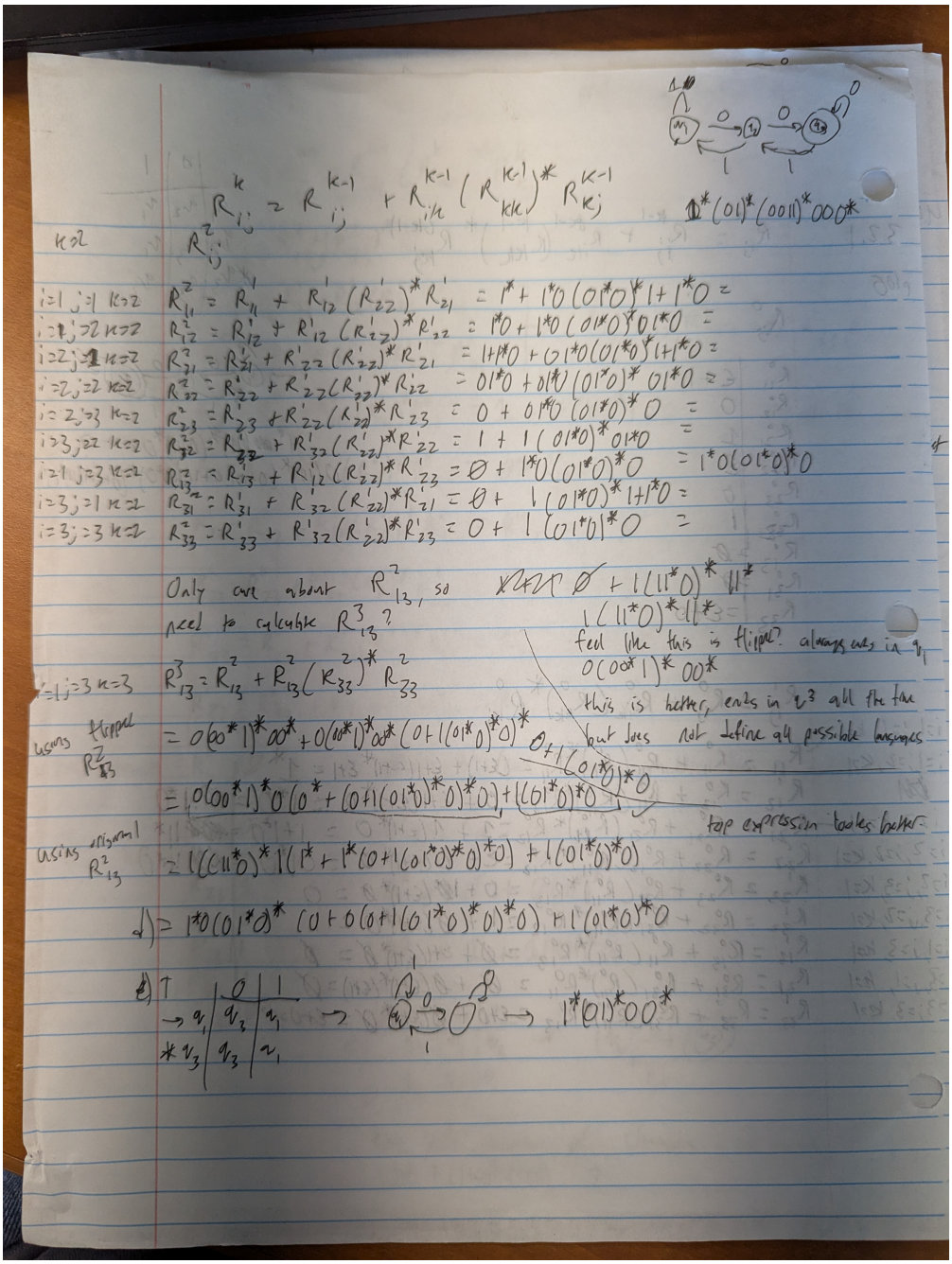

Figure 4: ITALC 3.2.1 Part 2

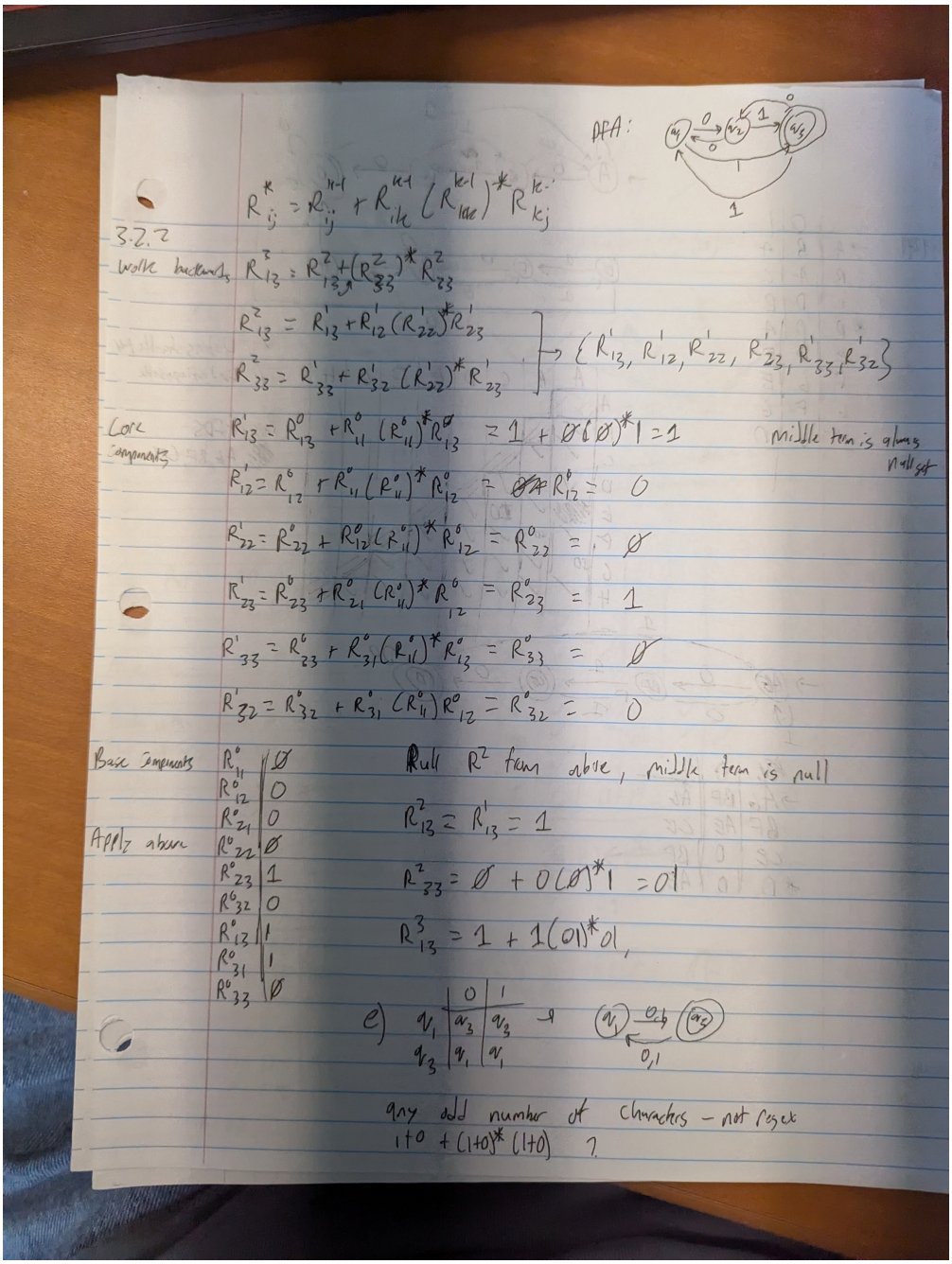

Figure 5: ITALC 3.2.2

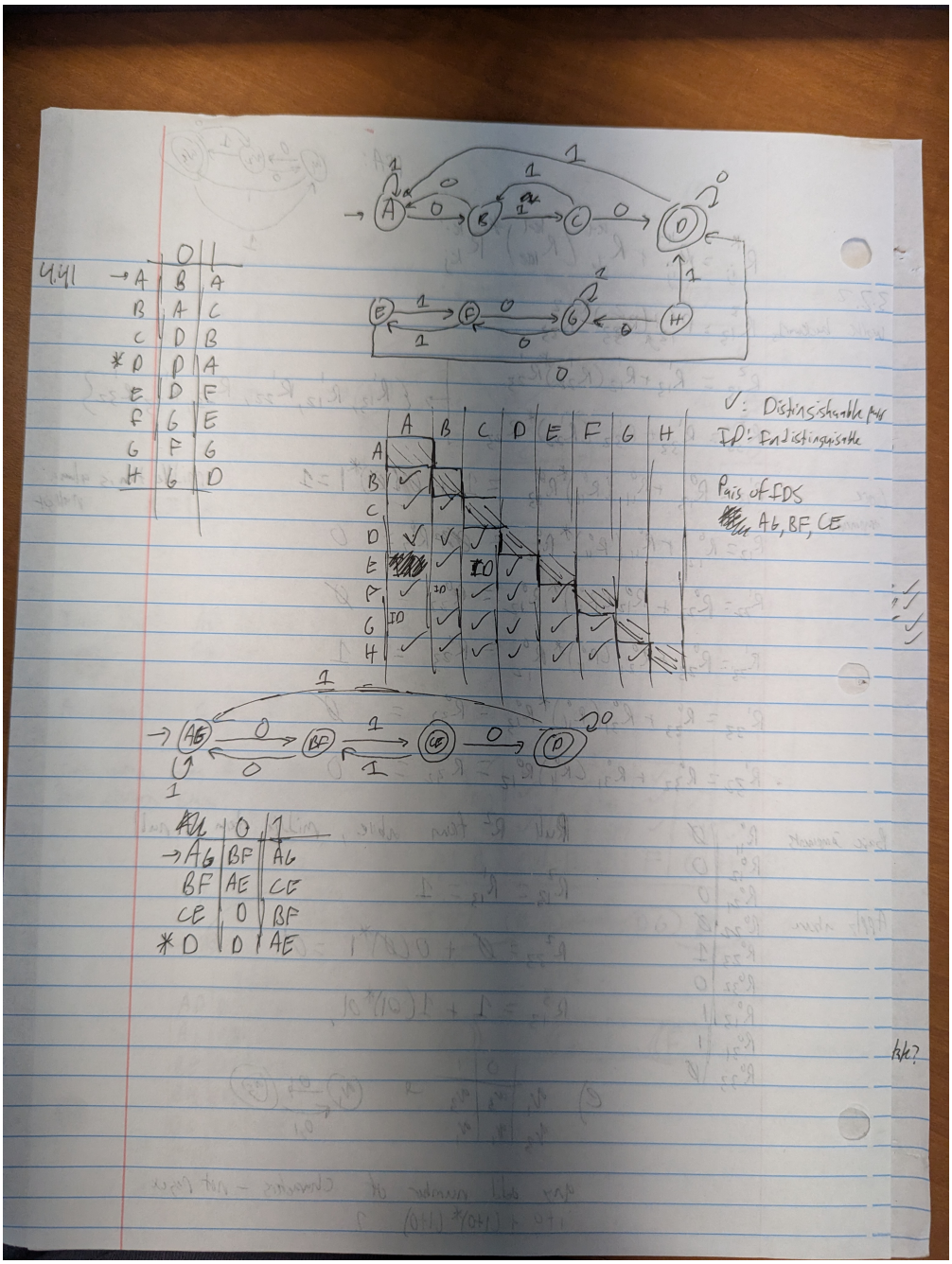

Figure 6: ITALC 4.4.1

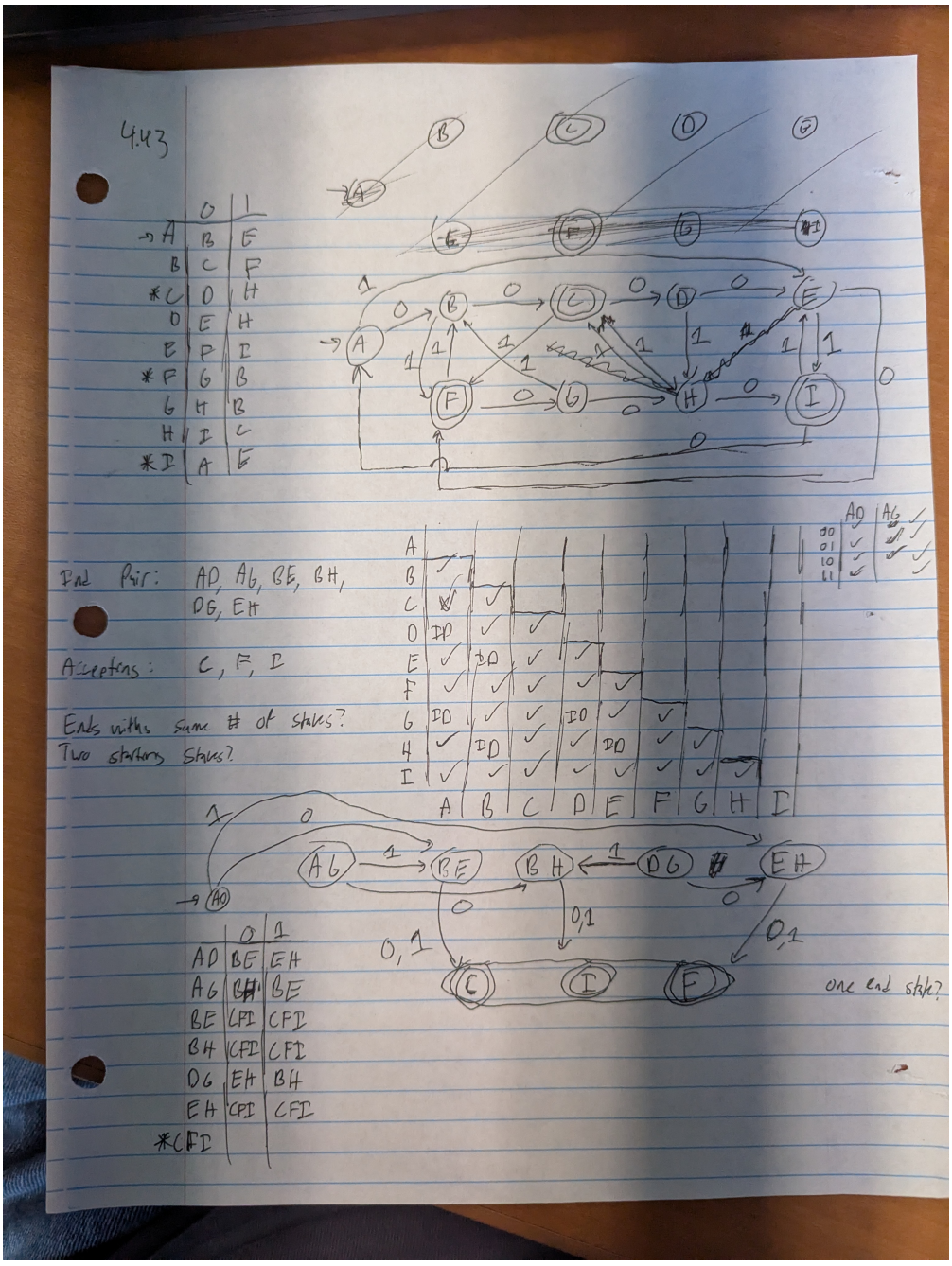

Figure 7: ITALC 4.4.2

3 Synthesis

4 Evidence of Participation

5 Conclusion

References

[BLA] Author, Title, Publisher, Year.