CPSC-406 Report

Zach Pratto
Chapman University

April 11, 2025

Abstract

Contents

1 Introduction **2**

2 Week by Week **2**
2.1 HW 1 . . . 2
2.1.1 Introduction to Automata Theory: Homework . . . 2
2.1.2 DFAs: Homework . . . 2
2.2 HW 2 . . . 3
2.2.1 Code from lab . . . 3
2.2.2 Programming with automata, exercise 4 . . . 5
2.2.3 Exercise 2.2.4 . . . 7
2.3 HW 3 . . . 8
2.3.1 Operations on automata: homework 1 and 2 . . . 8
2.3.2 Exercise 2.2.7 . . . 10
2.4 HW 4 . . . 11
2.4.1 Determinization: homework 1 and 2 . . . 11
2.5 HW 5 . . . 16
2.5.1 Exercise 3.2.1 . . . 16
2.5.2 Exercise 3.2.2 . . . 18
2.5.3 Exercise 4.4.1 . . . 19
2.5.4 Exercise 4.4.2 . . . 21

3 Synthesis **22**

4 Evidence of Participation **22**

5 Conclusion **22**

1 Introduction

2 Week by Week

2.1 HW 1

2.1.1 Introduction to Automata Theory: Homework

1. Parking machine accepts sequences like:

$$55555, \quad 555\ 10, \quad 55\ 10\ 5, \quad 5\ 10\ 55, \quad 10\ 555,$$
$$10\ 10\ 5, \quad 10\ 5\ 10, \quad 5\ 10\ 10$$

Accepted words have no more than two tens (10s).

$$\text{Regex:} \quad \text{pay}(\text{push} + \text{pay}^*) + \text{push}^*$$

2.1.2 DFAs: Homework

$$L_1 = \{x01y \mid \text{for all } x, y \text{ in } \Sigma^*\}$$

Does the word contain the substring 01?

$$L_2 = \{w \mid |w| = 2^n \text{ for some } n \in \mathbb{N}\}$$

Is the length of the word a power of 2?

$$L_3 = \{w \mid |w|_0 = |w|_1\}$$

Does the word have the same number of 0s and 1s?

$$w_1 = 10011 \Rightarrow L_1 : \text{yes}, \quad L_2 : \text{no}, \quad L_3 : \text{no}$$
$$w_2 = 100 \Rightarrow L_1 : \text{no}, \quad L_2 : \text{no}, \quad L_3 : \text{no}$$
$$w_3 = 10100100 \Rightarrow L_1 : \text{yes}, \quad L_2 : \text{yes}, \quad L_3 : \text{no}$$
$$w_4 = 1010011100 \Rightarrow L_1 : \text{yes}, \quad L_2 : \text{no}, \quad L_3 : \text{yes}$$
$$w_5 = 11110000 \Rightarrow L_1 : \text{no}, \quad L_2 : \text{yes}, \quad L_3 : \text{yes}$$

$$\text{Test words for a DFA:} \quad w_1 = 0010, \quad w_2 = 1101, \quad w_3 = 1100$$
$$\text{End states:} \quad w_1 \to q_1, \quad w_2 \to q_1, \quad w_3 \to q_2$$
$$\text{Observation:} \quad \text{All accepted words seem to require an 01 substring.}$$

Chapter 2.1 discusses how finite automata accomplish tasks and the problems that arise in their design. An example shows an automata schema for a customer, store, and bank in an electronic transaction. Initial

simple schemas fail to handle errors, leading to stateless situations. Fixing these issues allows running the automata in parallel, creating $1 \times 4 \times 7 = 28$ states. Arcs define transitions, and unreachable states are removed. This illustrates key automata principles.

Question: Generally, how much of the overall design for a given automata goes into handling "bad" inputs? It seems like designs can start fairly simple but grow complicated due to having to handle so many different scenarios, especially undesirable ones.

2.2 HW 2

2.2.1 Code from lab

dfa.py (modified)

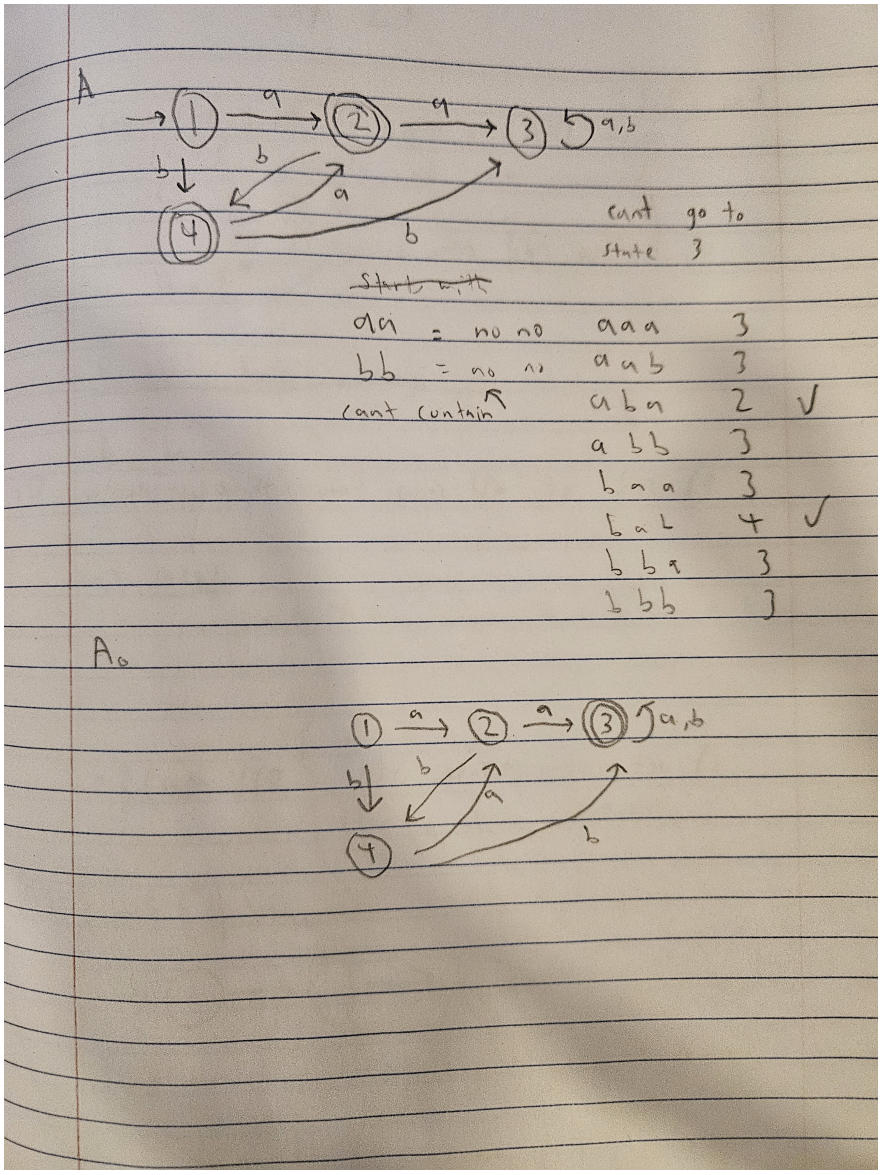
```
# a class for DFAs
# modify as needed
class DFA :

    # init the DFA
    def __init__(self, Q, Sigma, delta, q0, F) :
        self.Q = Q # set of states
        self.Sigma = Sigma # set of symbols
        self.delta = delta # transition function
        self.q0 = q0 # initial state
        self.F = F # final states

   # print the data of the DFA
    def __repr__(self) :
        return f"DFA({self.Q},\n\t{self.Sigma},\n\t{self.delta},\n\t{self.q0},\n\t{self.F})"

    # run the DFA on the word w
    # return if the word is accepted or not
    # modify as needed
    def run(self, w) :
        # run always starts at q0 starting state
        current_state = self.q0
        for symbol in w:
            if symbol not in self.Sigma:
                print(f"Error: Symbol '{symbol}' not in the alphabet {self.Sigma}.")
                return False

            current_state = self.delta.get((current_state, symbol))
            if current_state is None:
                return False
        return current_state in self.F

    def refuse(self) :
        new_F = []
        for state in self.Q :
            if state not in self.F :
                new_F.append(state)
        return DFA(self.Q, self.Sigma, self.delta, self.q0, set(new_F))
```

dfaex01.py (modified)

```
import dfa

# generate words for testing
def generate_words():
    words = []
    alphabet = ['a', 'b']
    for first in alphabet:
        for second in alphabet:
            for third in alphabet:
                words.append(first + second + third)
    return words

def __main__() :

    Q1 = {1, 2, 3, 4}
    Sigma1 = ['a', 'b']
    delta1 = {
        (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 = 1
    F1 = {3} # Only state 3 is accepting
    A1 = dfa.DFA(Q1, Sigma1, delta1, q0_1, F1)

    Q2 = {1, 2, 3}
    Sigma2 = ['a', 'b']
    delta2 = {
        (1, 'a'): 2,
        (1, 'b'): 1,
        (2, 'a'): 3,
        (2, 'b'): 1,
        (3, 'a'): 3,
        (3, 'b'): 1,
    }
    q0_2 = 1
    F2 = {3}
    A2 = dfa.DFA(Q2, Sigma2, delta2, q0_2, F2)

    words = generate_words()
    automata = [A1, A2]

    customWords = ["abbabaaabab", "bbbababba", "abbabaabab"]

    # test words on automata
    for X in automata:
         print(f"{X.__repr__()}")
         for w in words:
            print(f"{w}: {X.run(w)}")
         print("\n")
```

```
    # custom word test
    for X in automata:
        print(f"{X.__repr__()}")
        for w in customWords:
            print(f"{w}: {X.run(w)}")
        print("\n")

__main__()
```

2.2.2 Programming with automata, exercise 4



dfaex03.py (modified)

```
import dfa

def __main__() :

    Q1 = {1, 2, 3, 4}
    Sigma1 = ['a', 'b']
    delta1 = {
        (1, 'a'): 2,
        (1, 'b'): 4,
        (2, 'a'): 3,
        (2, 'b'): 4,
        (3, 'a'): 3,
        (3, 'b'): 3,
        (4, 'a'): 2,
        (4, 'b'): 3,
    }
    q0_1 = 1
    F1 = {2, 4}
    A = dfa.DFA(Q1, Sigma1, delta1, q0_1, F1)

    Q2 = {1, 2, 3, 4}
    Sigma2 = ['a', 'b']
    delta2 = {
        (1, 'a'): 2,
        (1, 'b'): 4,
        (2, 'a'): 3,
        (2, 'b'): 4,
        (3, 'a'): 3,
        (3, 'b'): 3,
        (4, 'a'): 2,
        (4, 'b'): 3,
    }
    q0_2 = 1
    F2 = {3} # Only state 3 is accepting
    A1 = dfa.DFA(Q2, Sigma2, delta2, q0_2, F2)
    #A1 should be equivalent to A0 if tests are correct
    #all that I did was swap the final, non initial states with non-final
    #non starting states, meaning in this case all that changes
    #is F containing 3 instead of 2, 4

    #testWord1 should be accepted by A and refused by A0
    #testWord2 should be accepted by A0 and refused by A
    testWords1 = ["aababaaba", "ababaab", "bbaababaa"]
    testWords2 = ["ababa", "bababa", "abababababab"]

    A0 = A.refuse()

    testAutomata = [A, A1, A0]
    print("Test words 1\n")
    for automata in testAutomata:
        print(f"{automata.__repr__()}")
        for w in testWords1:
            print(f"{w}: {automata.run(w)}")
        print("\n")
```

```
    print("Test words 2\n")
    for automata in testAutomata:
        print(f"{automata.__repr__()}")
        for w in testWords2:
            print(f"{w}: {automata.run(w)}")
        print("\n")

__main__()
```

The modifications made to dfa.py were to add the run and refuse methods. The run method works by essentially telling us the next state that would be in after the last symbol is seen by the function. So for A1, if 2 is the last state seen, and the incoming symbol is b, then the final state returned will be state 3.

In the refuse method works on an instance of the DFA object that calls it, I start with an empty set of states, and iterate over the set of all states of the DFA. Then, that state is appended to the empty set of states as the function checks if the states in the set F of final states. Basically, the goal to make the compliment of A was to swap the middle states with final states, meaning the opposite words are accepted.

2.2.3 Exercise 2.2.4

1. $$L_1 = \{w \mid w \text{ ends with } 00, \text{ where } w = x00, \text{ for some string } x \text{ consisting of only 0's and 1's}\}$$

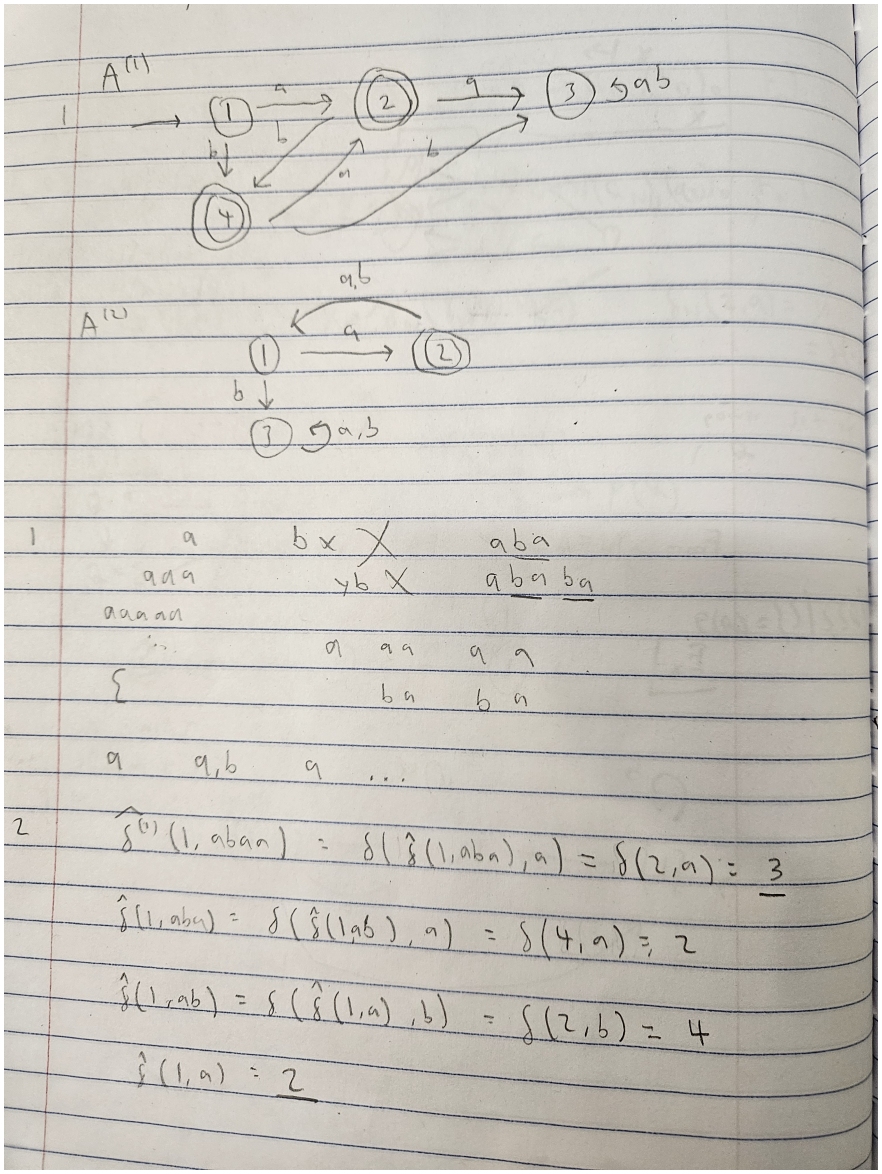
2. $$L_2 = \{w \mid w \text{ contains 000 as a substring, meaning } w = x000y, 000y, \text{ or } x000, \text{ where } x, y \text{ are strings consisting of only 0's and 1's}\}$$

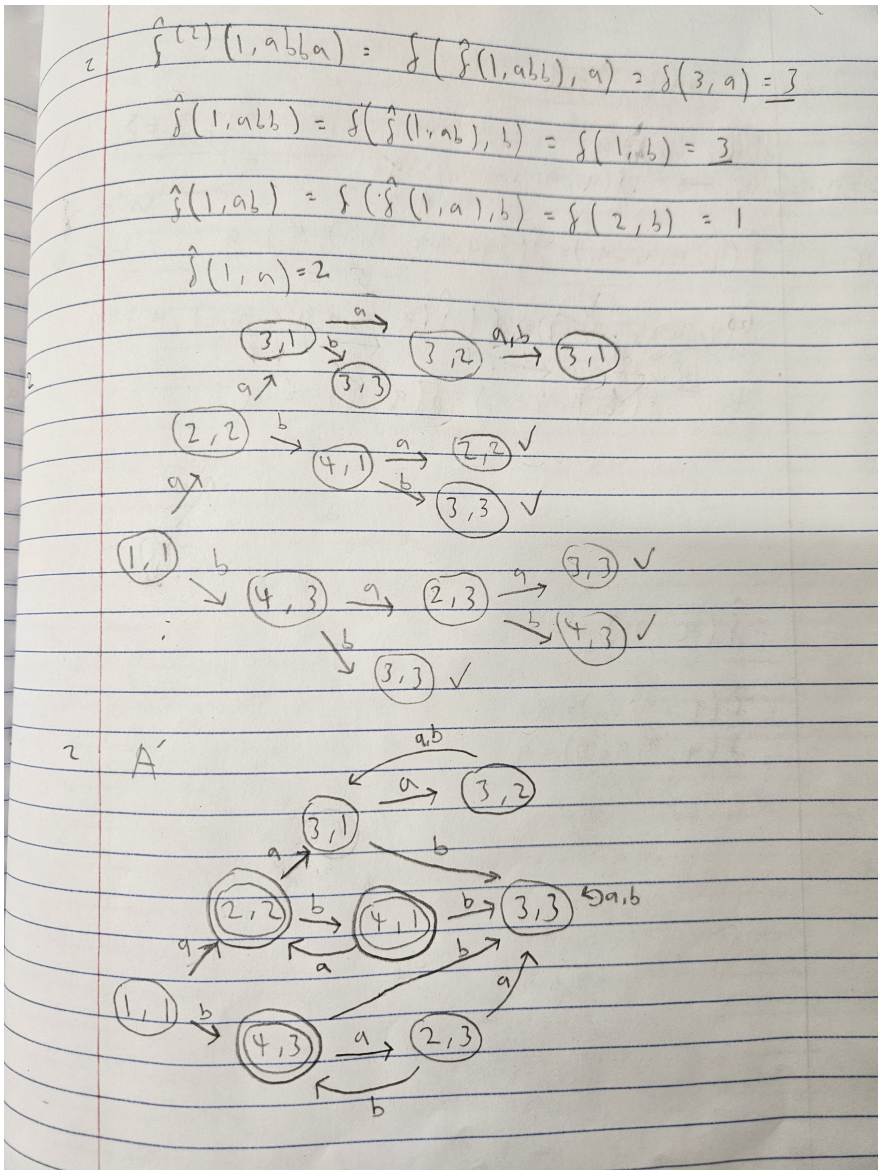
3. $$L_3 = \{w \mid w \text{ contains 011 as a substring, meaning } w = x011y, \text{ where } x, y \text{ are strings consisting of only 0's and 1's}\}$$

Question: Aside from tables and diagrams, could there be another niche method of representing DFAs that would be useful for some purpose?

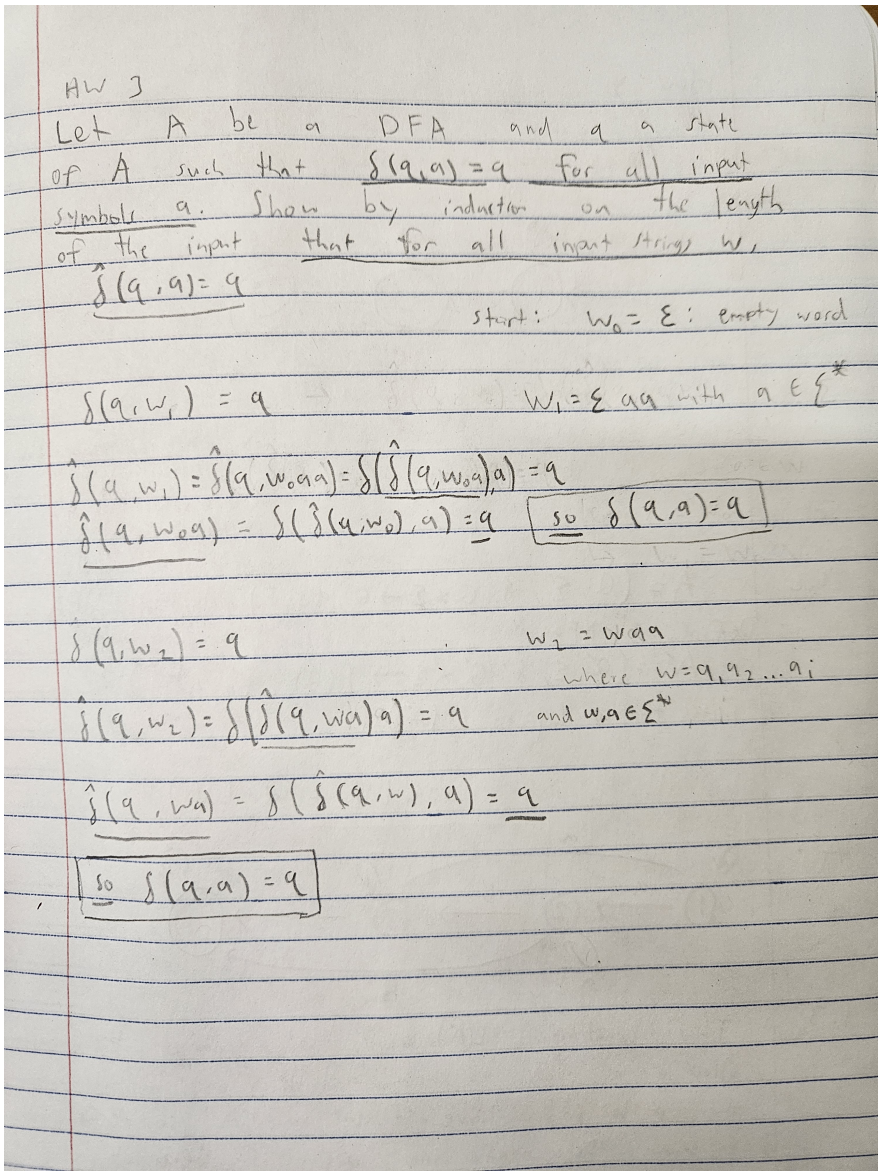
2.3 HW 3

2.3.1 Operations on automata: homework 1 and 2




2 $\hat{\delta}^{(2)}(1,abba) = \delta(\hat{\delta}(1,abb),a) = \delta(3,a) = 3$
$\hat{\delta}(1,abb) = \delta(\hat{\delta}(1,ab),b) = \delta(1,b) = 3$
$\hat{\delta}(1,ab) = \delta(\hat{\delta}(1,a),b) = \delta(2,b) = 1$
$\hat{\delta}(1,a) = 2$
3,1
3,2
3,3
2,2
4,1
1,1
4,3
2,3
2 A'
a,b
5a,b

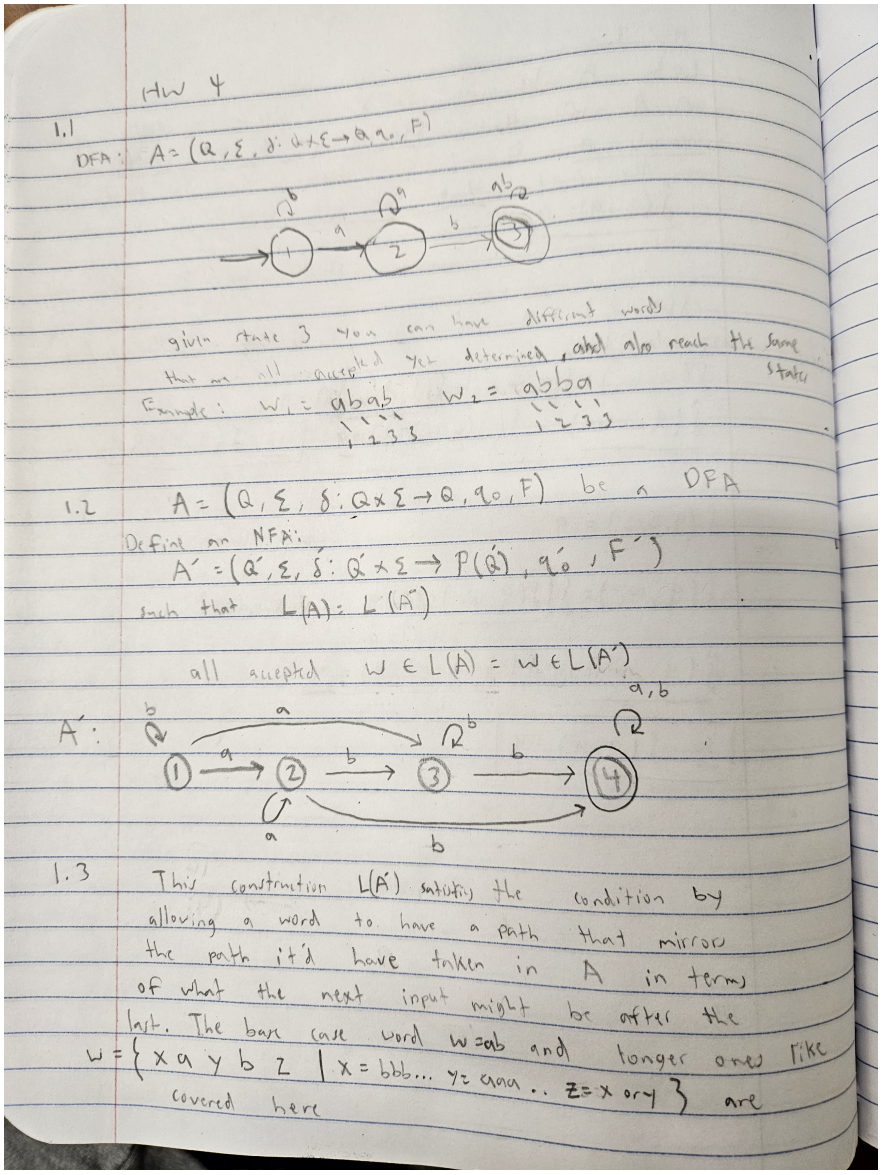
2.3.2 Exercise 2.2.7



Question: Can two different NFAs recognize the same language but process words in fundamentally different ways?

2.4 HW 4

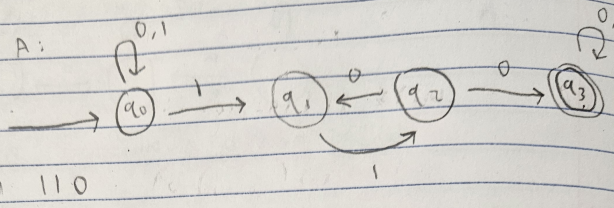
2.4.1 Determinization: homework 1 and 2



HW 4

2.1 Describe $L(A)$

NFA A:



accepted 110

1111...0 ..111..000..0101

110, 11010101...10

accepted words must contain 110 or contain 11 followed by 01...

2.2 $Q: q_0, q_1, q_2, q_3$

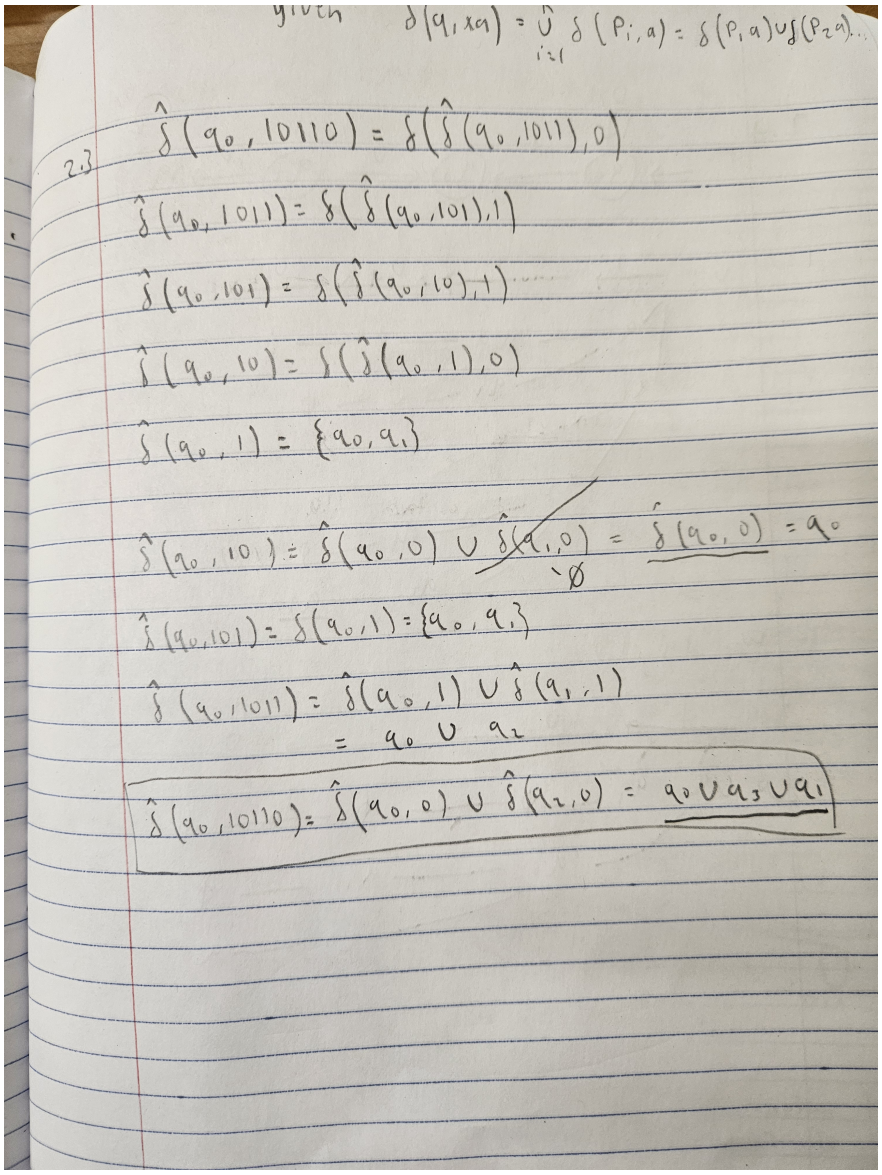
$\Sigma: \{0,1\}$

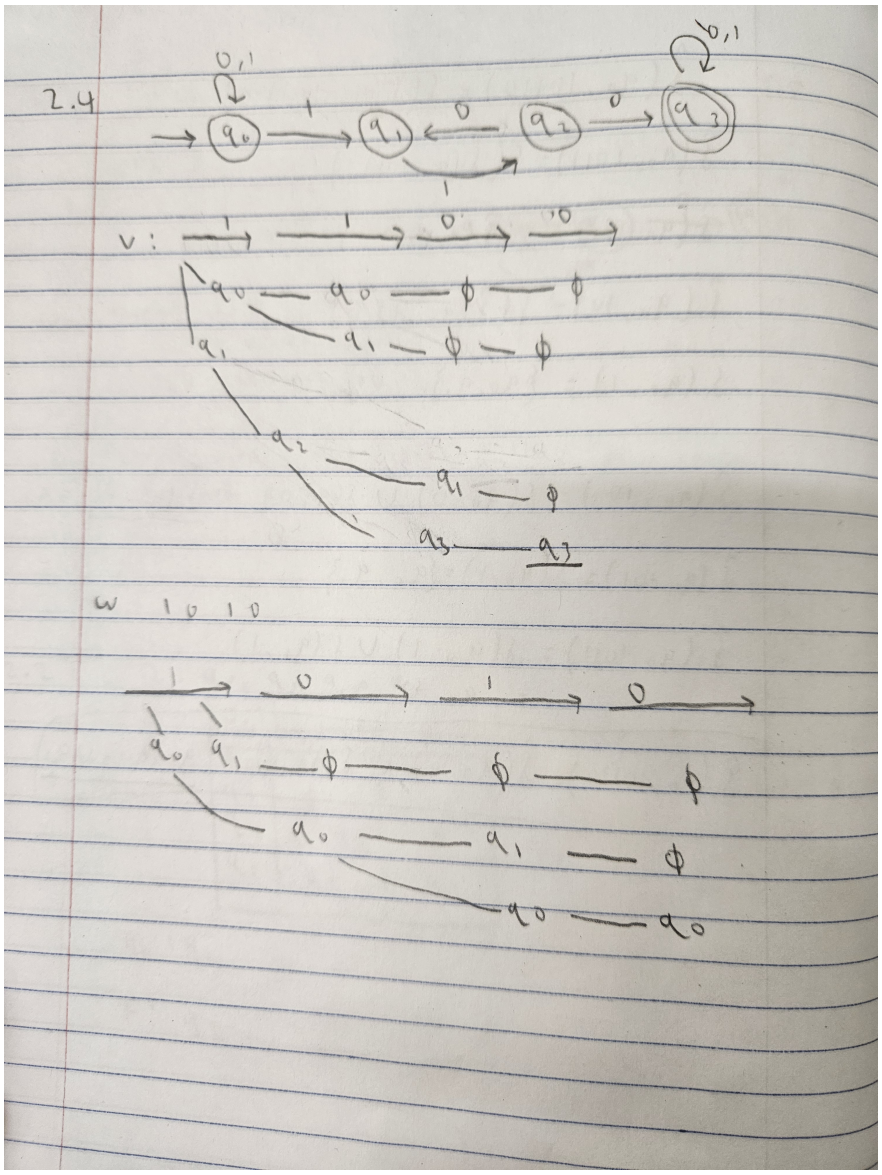
δ:

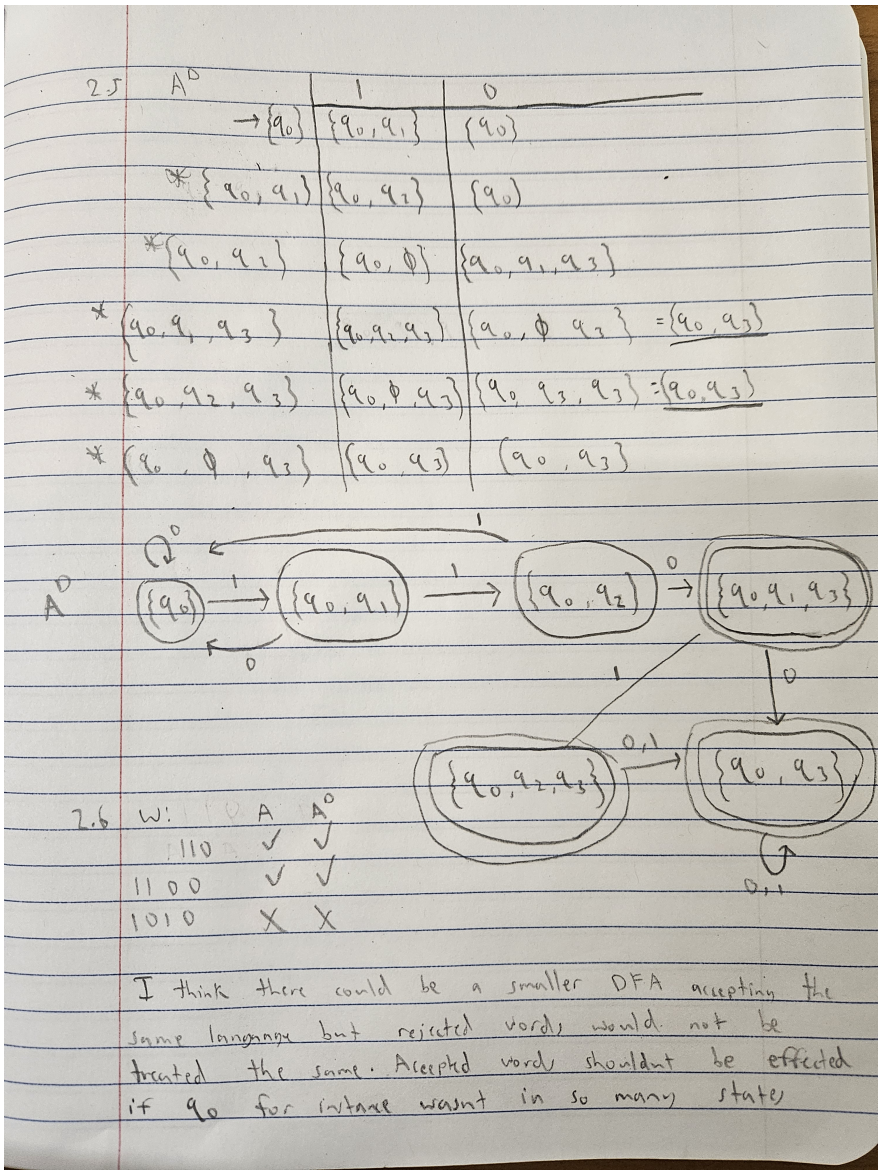
	0	1
q_0	q_0	q_0, q_1
q_1	ϕ	q_2
q_2	q_1, q_3	ϕ
q_3	q_3	q_3

$q_0: q_0$

$F: q_3$


given $\delta(q, xa) = \bigcup_{i=1}^{n} \delta(p_i, a) = \delta(p_1, a) \cup \delta(p_2, a) \ldots$
2.3
$\hat{\delta}(q_0, 10110) = \delta(\hat{\delta}(q_0, 1011), 0)$
$\hat{\delta}(q_0, 1011) = \delta(\hat{\delta}(q_0, 101), 1)$
$\hat{\delta}(q_0, 101) = \delta(\hat{\delta}(q_0, 10), 1)$
$\hat{\delta}(q_0, 10) = \delta(\hat{\delta}(q_0, 1), 0)$
$\hat{\delta}(q_0, 1) = \{q_0, q_1\}$
$\hat{\delta}(q_0, 10) = \hat{\delta}(q_0, 0) \cup \hat{\delta}(q_1, 0) = \hat{\delta}(q_0, 0) = q_0$
$\emptyset$
$\hat{\delta}(q_0, 101) = \delta(q_0, 1) = \{q_0, q_1\}$
$\hat{\delta}(q_0, 1011) = \hat{\delta}(q_0, 1) \cup \hat{\delta}(q_1, 1)$
$= q_0 \cup q_2$
$\hat{\delta}(q_0, 10110) = \hat{\delta}(q_0, 0) \cup \hat{\delta}(q_2, 0) = q_0 \cup q_3 \cup q_1$

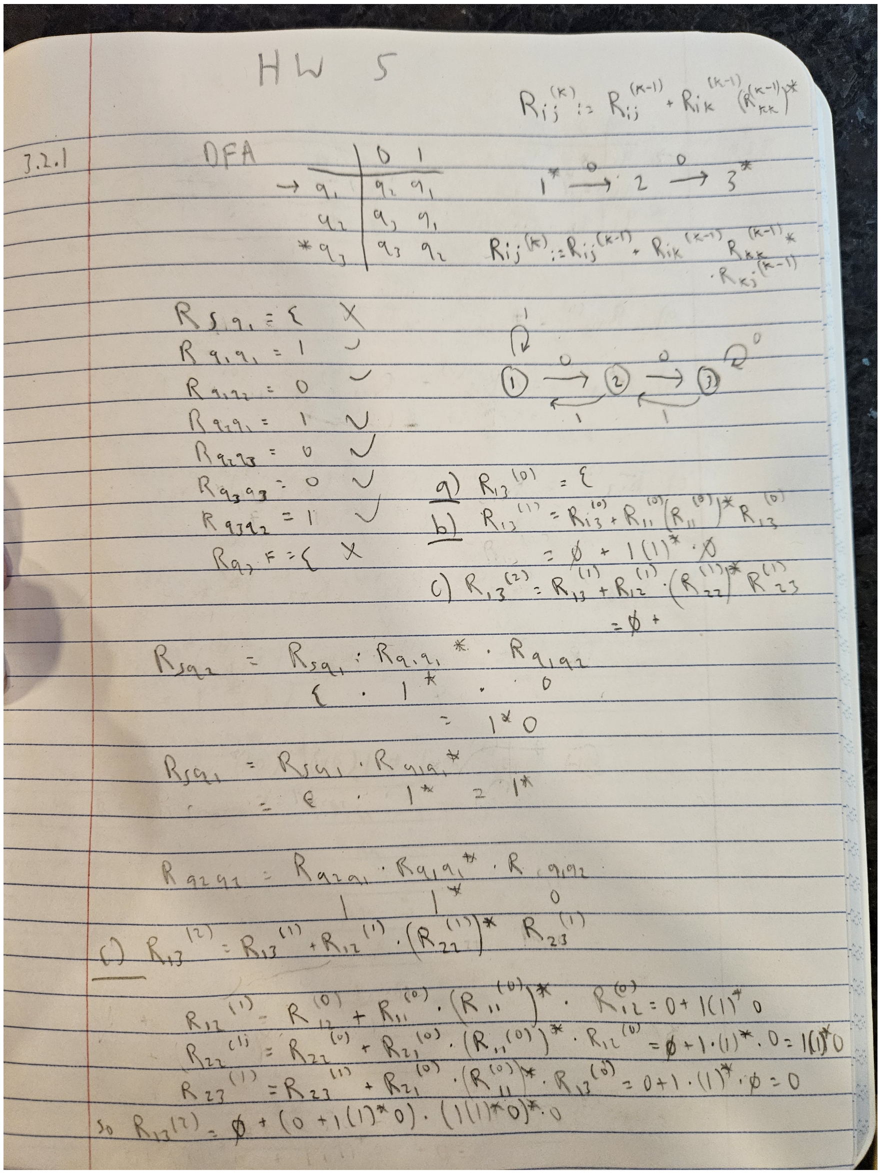

2.4
0,1
0,1
1
0
0
1
v: 1 1 0 0
q0 q0 ϕ ϕ
q1 q1 ϕ ϕ
q2 q1 ϕ
q3 q3
w 1 0 1 0
1 0 1 0
q0 q1 ϕ ϕ ϕ
q0 q1 ϕ
q0 q0



Question: What patterns can we observe in potential real world applications for automata that indicate strongly or weakly which type of automatom is most useful?

2.5 HW 5

2.5.1 Exercise 3.2.1



HW 5

$R_{ij}^{(k)} := R_{ij}^{(k-1)} + R_{ik}^{(k-1)} (R_{kk}^{(k-1)})^*$

3.2.1 DFA

	0	1
$\to q_1$	q_2	q_1
q_2	q_3	q_1
$*q_3$	q_3	q_2

$1^* \xrightarrow{0} 2 \xrightarrow{0} 3^*$

$R_{ij}^{(k)} := R_{ij}^{(k-1)} + R_{ik}^{(k-1)} R_{kk}^{(k-1)*} \cdot R_{kj}^{(k-1)}$

$R_{sq_1} = \epsilon$ X

$R_{q_1q_1} = 1$

$R_{q_1q_2} = 0$

$R_{q_2q_1} = 1$

$R_{q_2q_3} = 0$

$R_{q_3q_3} = 0$

$R_{q_3q_2} = 1$

$R_{q_3F} = \epsilon$ X

a) $R_{13}^{(0)} = \epsilon$

b) $R_{13}^{(1)} = R_{13}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{13}^{(0)}$

$= \emptyset + 1(1)^* \cdot \emptyset$

c) $R_{13}^{(2)} = R_{13}^{(1)} + R_{12}^{(1)} \cdot (R_{22}^{(1)})^* R_{23}^{(1)}$

$= \emptyset +$

$R_{sq_2} = R_{sq_1} \cdot R_{q_1q_1}^* \cdot R_{q_1q_2}$

$\epsilon \cdot 1^* \cdot 0$

$= 1^* 0$

$R_{sq_1} = R_{sq_1} \cdot R_{q_1q_1}^*$

$= \epsilon \cdot 1^* = 1^*$

$R_{q_2q_1} = R_{q_2q_1} \cdot R_{q_1q_1}^* \cdot R_{q_1q_2}$

$1 \quad 1^* \quad 0$

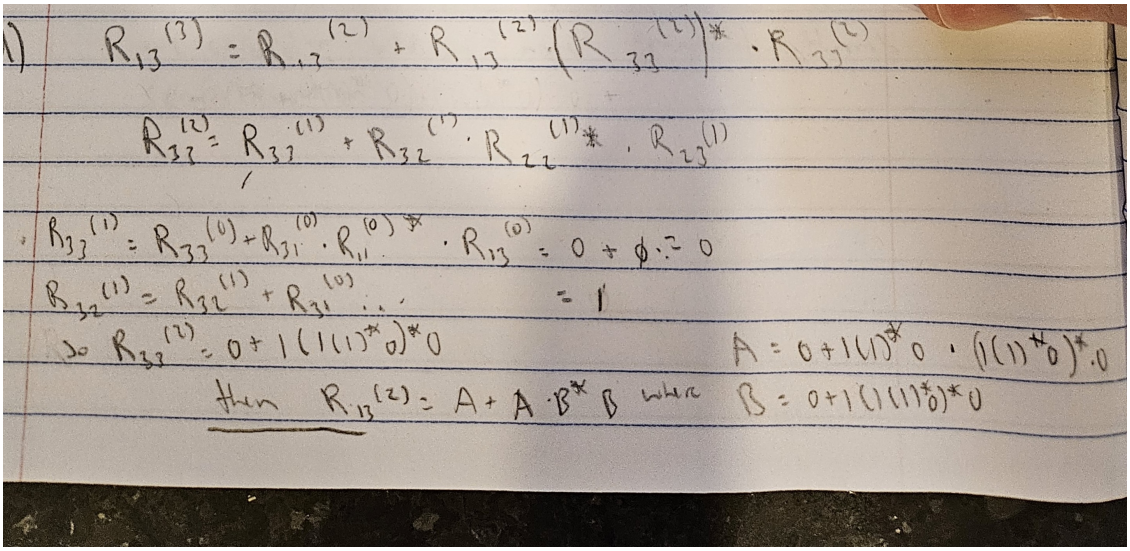
c) $R_{13}^{(2)} = R_{13}^{(1)} + R_{12}^{(1)} \cdot (R_{22}^{(1)})^* R_{23}^{(1)}$

$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)} \cdot (R_{11}^{(0)})^* \cdot R_{12}^{(0)} = 0 + 1(1)^* 0$

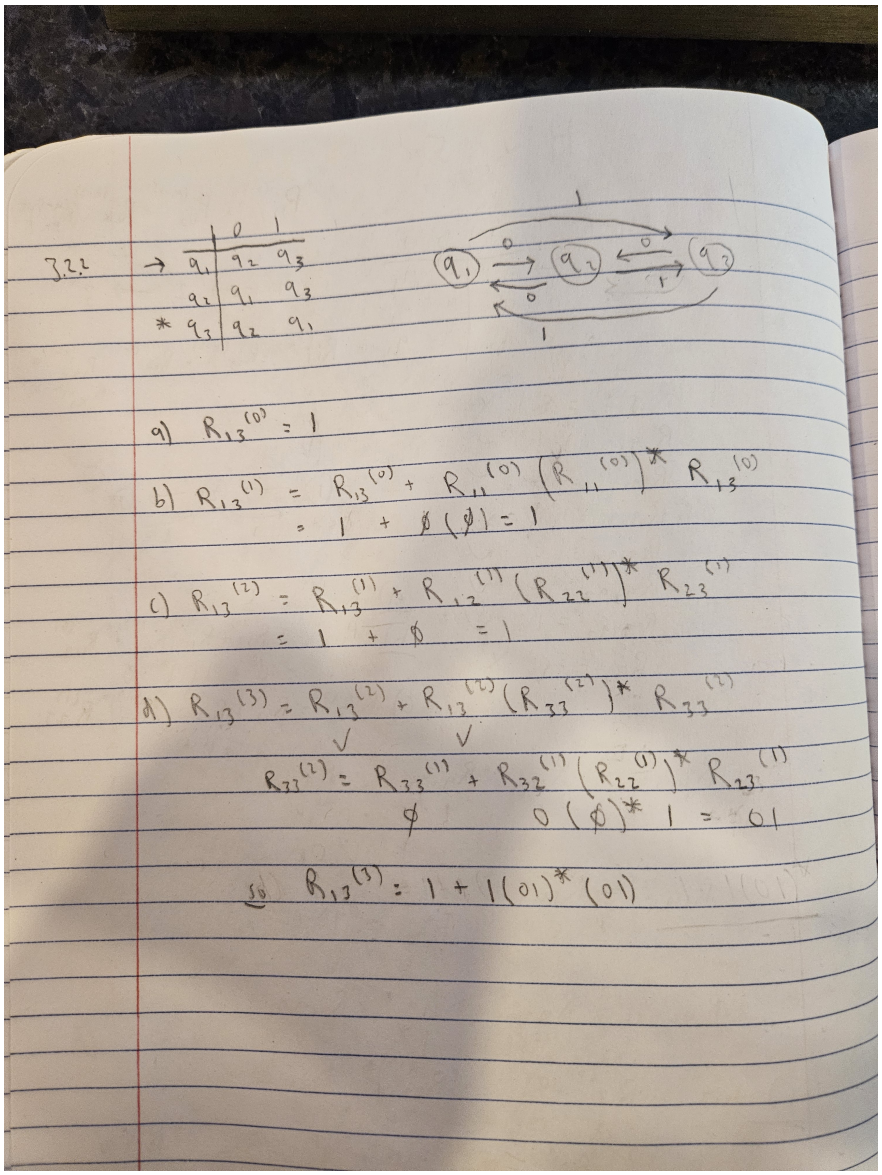
$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} \cdot (R_{11}^{(0)})^* \cdot R_{12}^{(0)} = \emptyset + 1 \cdot (1)^* \cdot 0 = 1(1)^* 0$

$R_{23}^{(1)} = R_{23}^{(1)} + R_{21}^{(0)} \cdot (R_{11}^{(0)})^* \cdot R_{13}^{(0)} = 0 + 1 \cdot (1)^* \cdot \emptyset = 0$

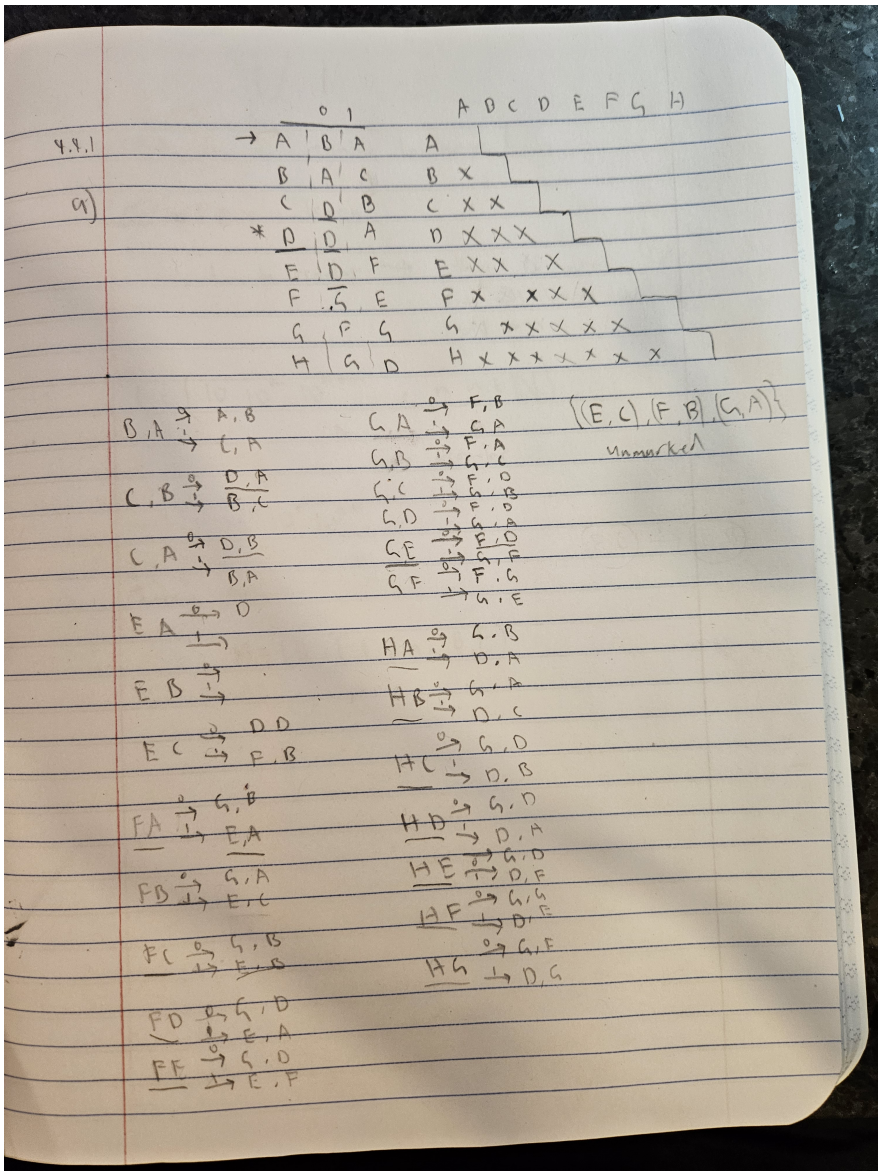
so $R_{13}^{(2)} = \emptyset + (0 + 1(1)^* 0) \cdot (1(1)^* 0)^* \cdot 0$

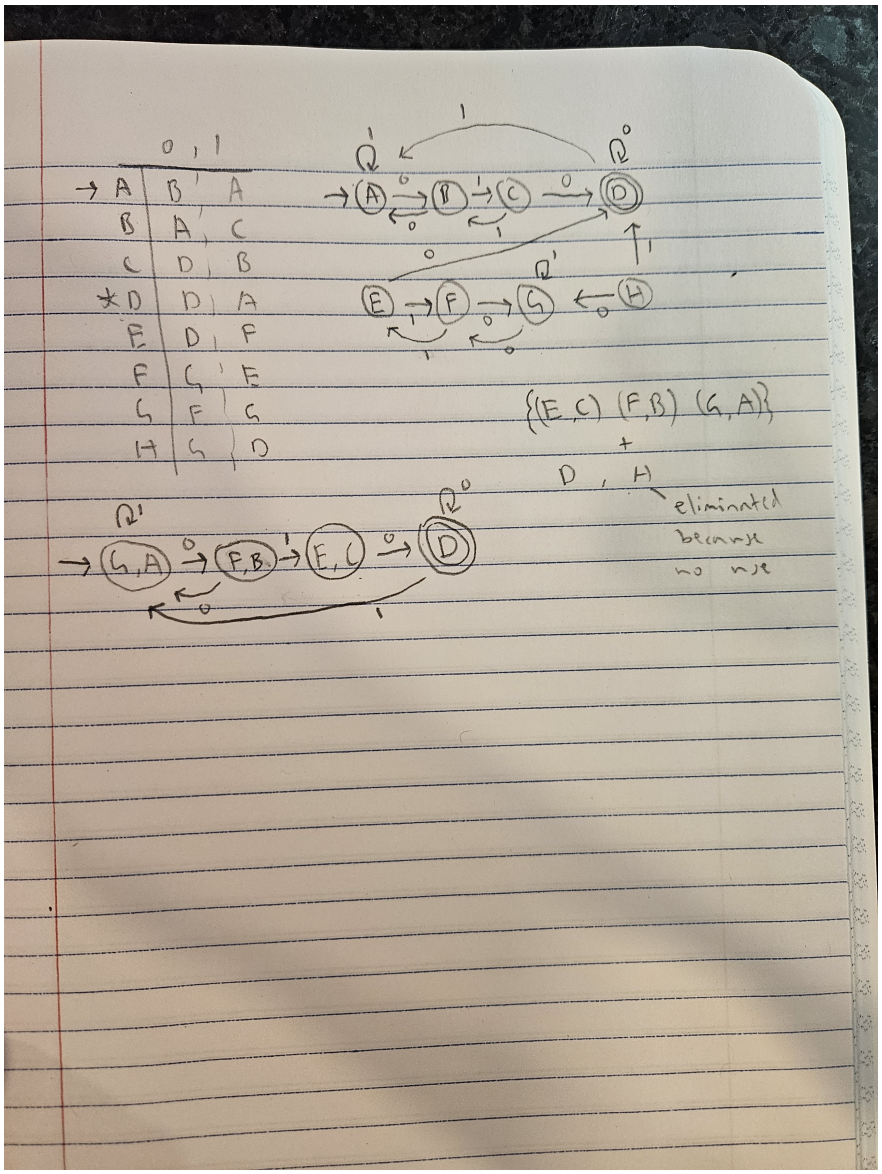

1) $R_{13}^{(3)} = R_{13}^{(2)} + R_{13}^{(2)} \cdot (R_{33}^{(2)})^* \cdot R_{33}^{(2)}$
$R_{33}^{(2)} = R_{33}^{(1)} + R_{32}^{(1)} \cdot R_{22}^{(1)*} \cdot R_{23}^{(1)}$
$R_{33}^{(1)} = R_{33}^{(0)} + R_{31}^{(0)} \cdot R_{11}^{(0)*} \cdot R_{13}^{(0)} = 0 + \phi \therefore = 0$
$R_{32}^{(1)} = R_{32}^{(1)} + R_{31}^{(0)} \ldots = 1$
$\therefore R_{33}^{(2)} = 0 + 1(1(1)^* 0)^* 0$
$A = 0 + 1(1)^* 0 \cdot (1(1)^* 0)^* \cdot 0$
Then $R_{13}^{(2)} = A + A \cdot B^* B$ where $B = 0 + 1(1(1)^* 0)^* 0$

2.5.2 Exercise 3.2.2

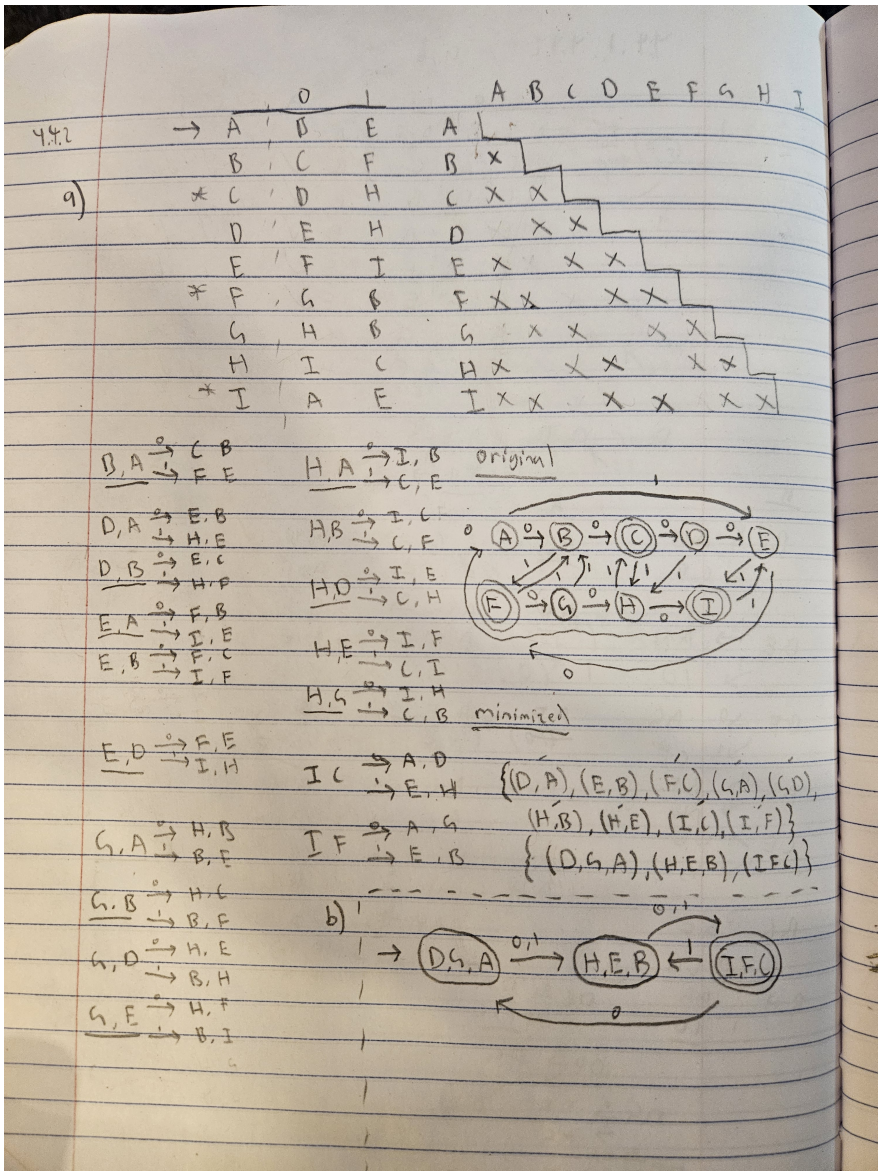


2.5.3 Exercise 4.4.1




0 , 1
→ A | B , A
B | A , C
C | D , B
*D | D , A
E | D , F
F | G , E
G | F , G
H | G , D
Ω'
Ω°
A
B
C
D
E
F
G
H
{(E,C) (F,B) (G,A)}
+
D , H
eliminated
because
no use
(G,A)
(F,B)
(E,C)
(D)

2.5.4 Exercise 4.4.2



Question: Is there a more compact way to display the same information as in a filling table once an automata reaches a certain size , or are other methods better/worse at being compact depending on number of accepting states or other variables?

3 Synthesis

4 Evidence of Participation

5 Conclusion

References

[BLA] Author, Title, Publisher, Year.