Finite Automata 2.1 Summary

Zach Pratto

March 10, 2025

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.

Week 3 .py Labs

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__()
```

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 = ["aabababa", "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.

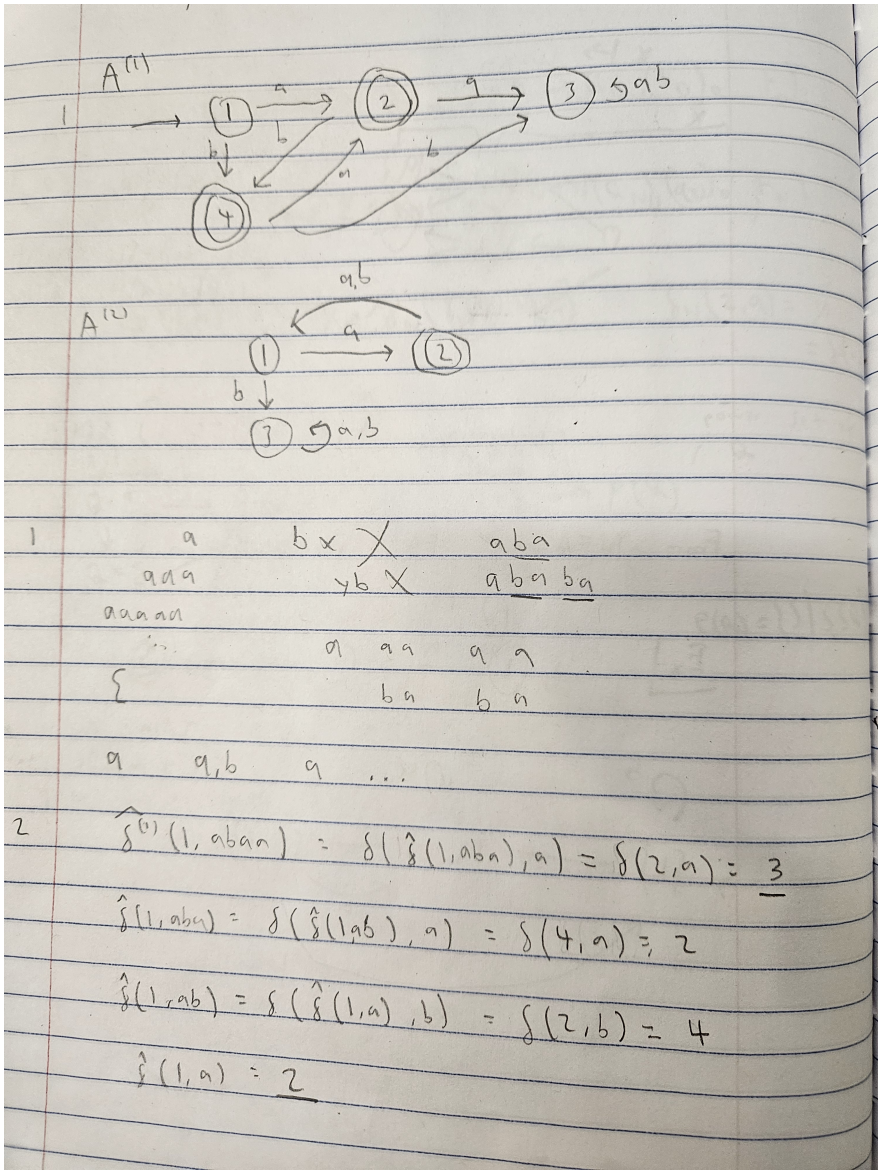
Exercise 2.2.4

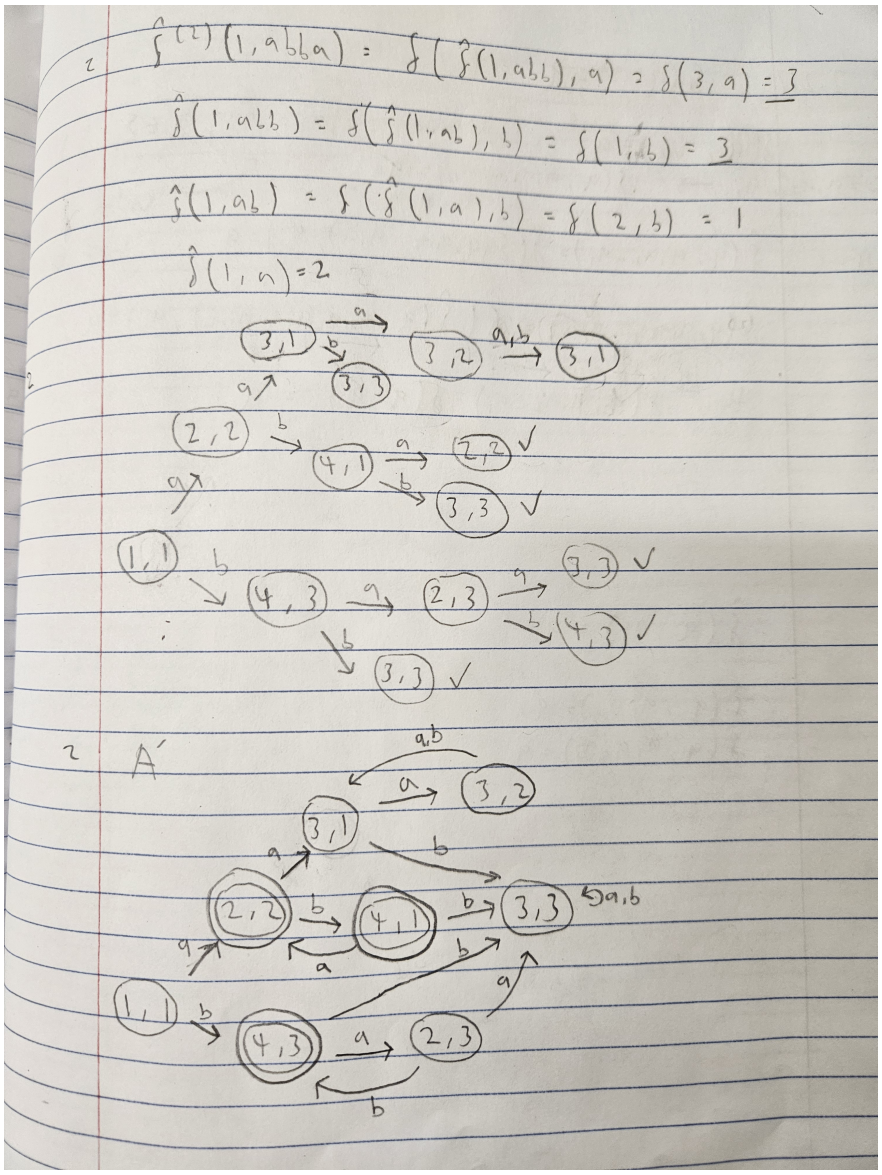
1. $$L_1 = \{w \mid w \text{ ends with } 00, \text{ where } w = x00 \text{ for some string } x \text{ consisting of only 0's and 1's}\}$$

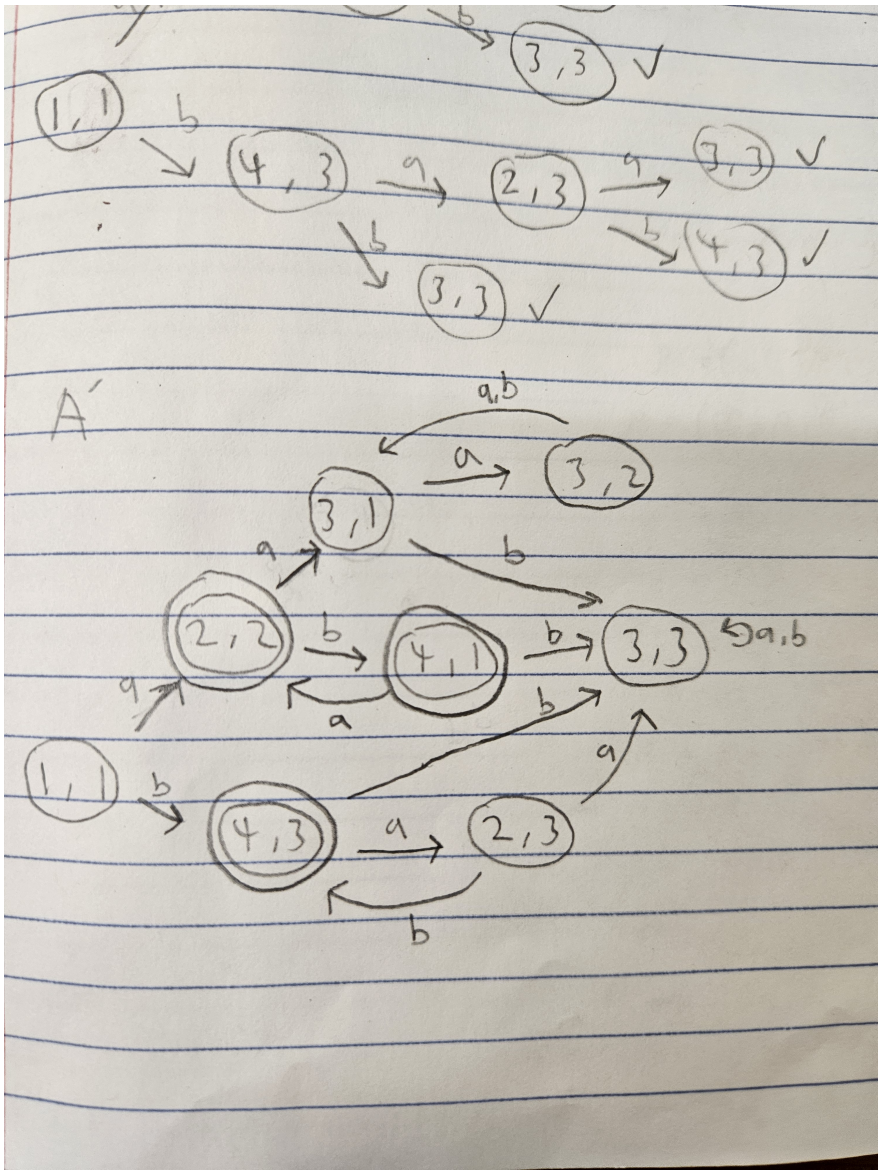
2. $$L_2 = \{w \mid w \text{ contains } 000 \text{ as a substring, meaning } w = x000y, 000y, \\ \text{or } x000, \text{ where } x, y \text{ are strings consisting of only 0's and 1's}\}$$

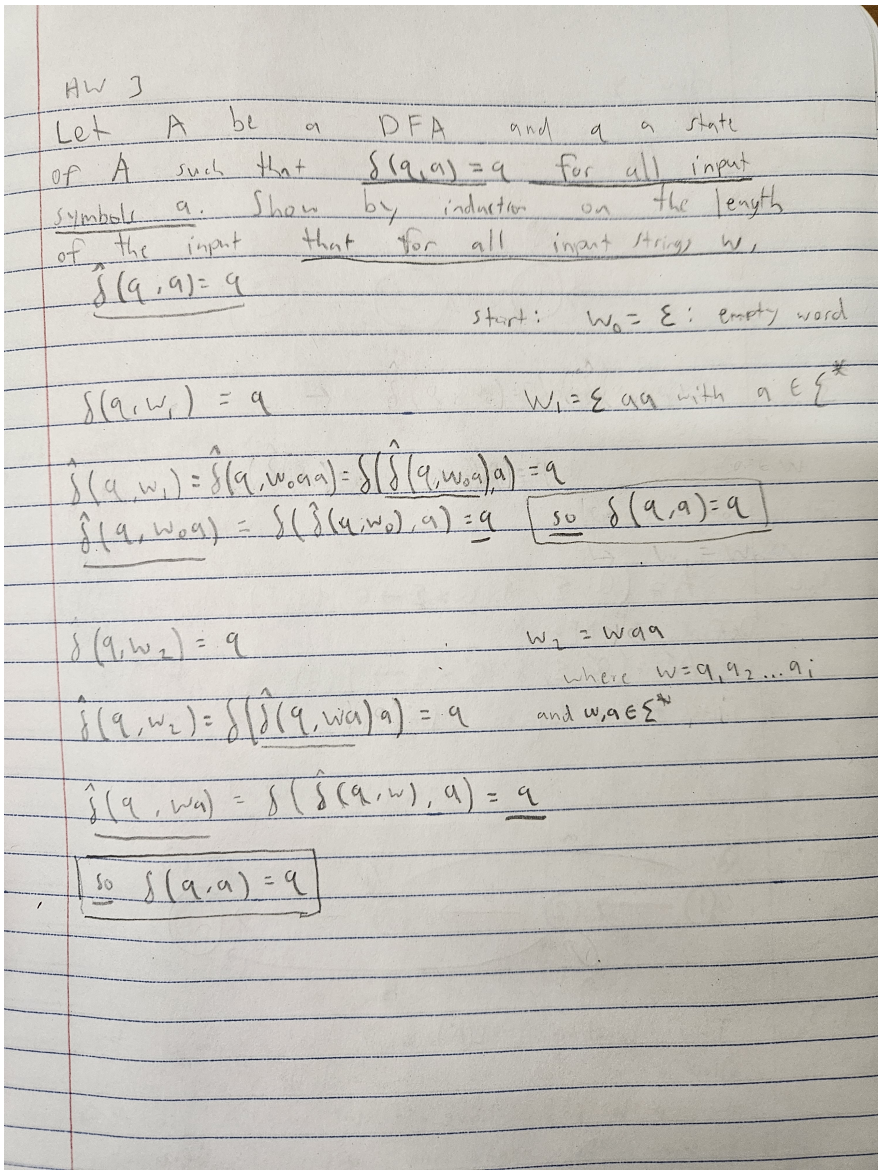
3. $L_3 = \{w \mid w \text{ contains } 011 \text{ as a substring, meaning } w = x011y, \text{ where } x, y \text{ are strings consisting of only 0's and 1's}\}$

HW 3

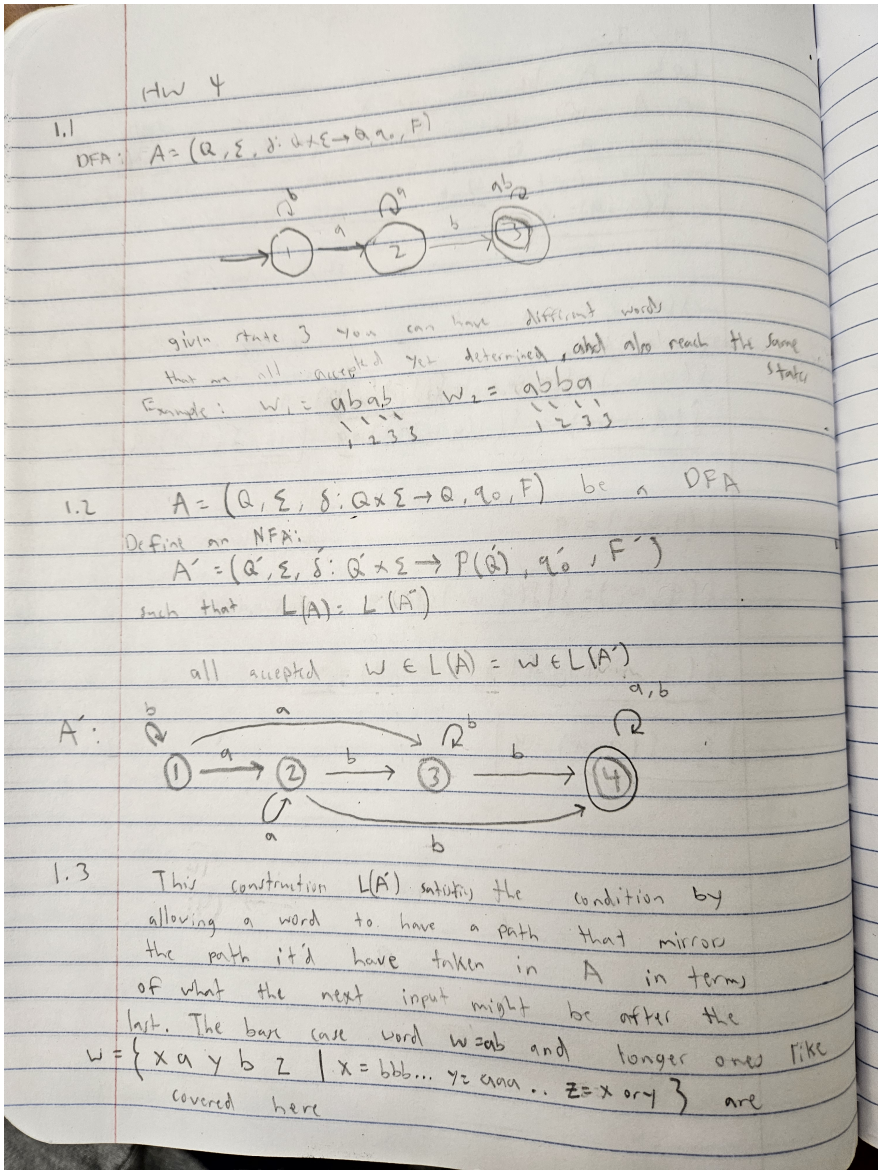

A(1)
1
2
3
4
a
b
a,b
A(2)
1
2
3
a,b
a
b x
aba
aaa
yb x
aba ba
aaaaa
a aa a a
ba b a
a a,b a ...
2
δ̂(1, abaa) = δ(δ̂(1, aba), a) = δ(2, a) = 3
δ̂(1, aba) = δ(δ̂(1, ab), a) = δ(4, a) = 2
δ̂(1, ab) = δ(δ̂(1, a), b) = δ(2, b) = 4
δ̂(1, a) = 2


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


3,3 ✓
1,1
b
4,3
a
2,3
a
3,3 ✓
b
4,3 ✓
b
3,3 ✓
A'
a,b
a
3,2
3,1
a
b
2,2
b
4,1
b
3,3
a,b
a
a
b
1,1
b
4,3
a
2,3
a
b


HW 3
Let A be a DFA and q a state of A such that $\delta(q,a) = q$ for all input symbols a. Show by induction on the length of the input that for all input strings w, $\hat{\delta}(q,a) = q$
start: $w_0 = \varepsilon$; empty word
$\delta(q,w_1) = q$ $w_1 = \varepsilon aa$ with $a \in \Sigma^*$
$\hat{\delta}(q,w_1) = \hat{\delta}(q,w_0aa) = \delta(\hat{\delta}(q,w_0a),a) = q$
$\hat{\delta}(q,w_0aa) = \delta(\hat{\delta}(q,w_0),a) = q$ so $\delta(q,a) = q$
$\delta(q,w_2) = q$ $w_2 = waa$
where $w = a_1a_2 \ldots a_i$
$\hat{\delta}(q,w_2) = \delta(\hat{\delta}(q,wa)a) = q$ and $w,a \in \Sigma^*$
$\hat{\delta}(q,wa) = \delta(\hat{\delta}(q,w),a) = q$
so $\delta(q,a) = q$

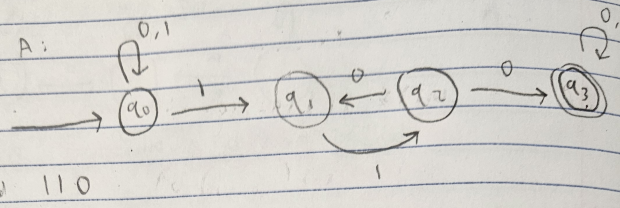
HW 4


HW 4
1.1
DFA: A = (Q, Σ, δ: Q×Σ→Q, q0, F)
b
a
a
b
ab
1
2
3
given state 3 you can have different words that are all accepted yet determined, and also reach the same states
Example: w1 = abab
w2 = abba
1.2
A = (Q, Σ, δ: Q×Σ→Q, q0, F) be a DFA
Define an NFA:
A' = (Q', Σ, δ': Q'×Σ→P(Q'), q0', F')
such that L(A) = L(A')
all accepted w ∈ L(A) = w ∈ L(A')
A':
a, b
1.3
This construction L(A') satisfies the condition by allowing a word to have a path that mirrors the path it'd have taken in A in terms of what the next input might be after the last. The base case word w=ab and longer ones like w = { x a y b z | x = bbb... y = anan.. z = x or y } are covered here

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 $Q: q_0, q_1, q_2, q_3$

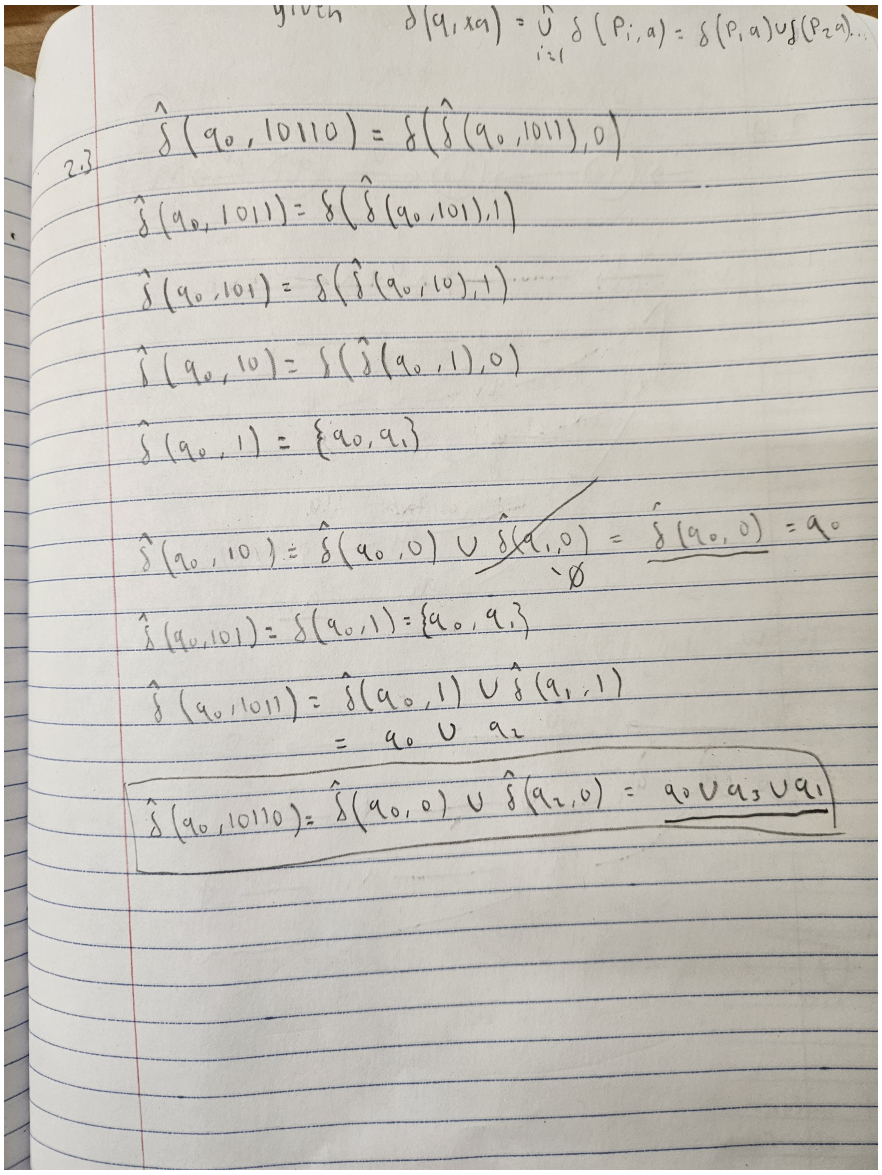
$\Sigma: \{0,1\}$

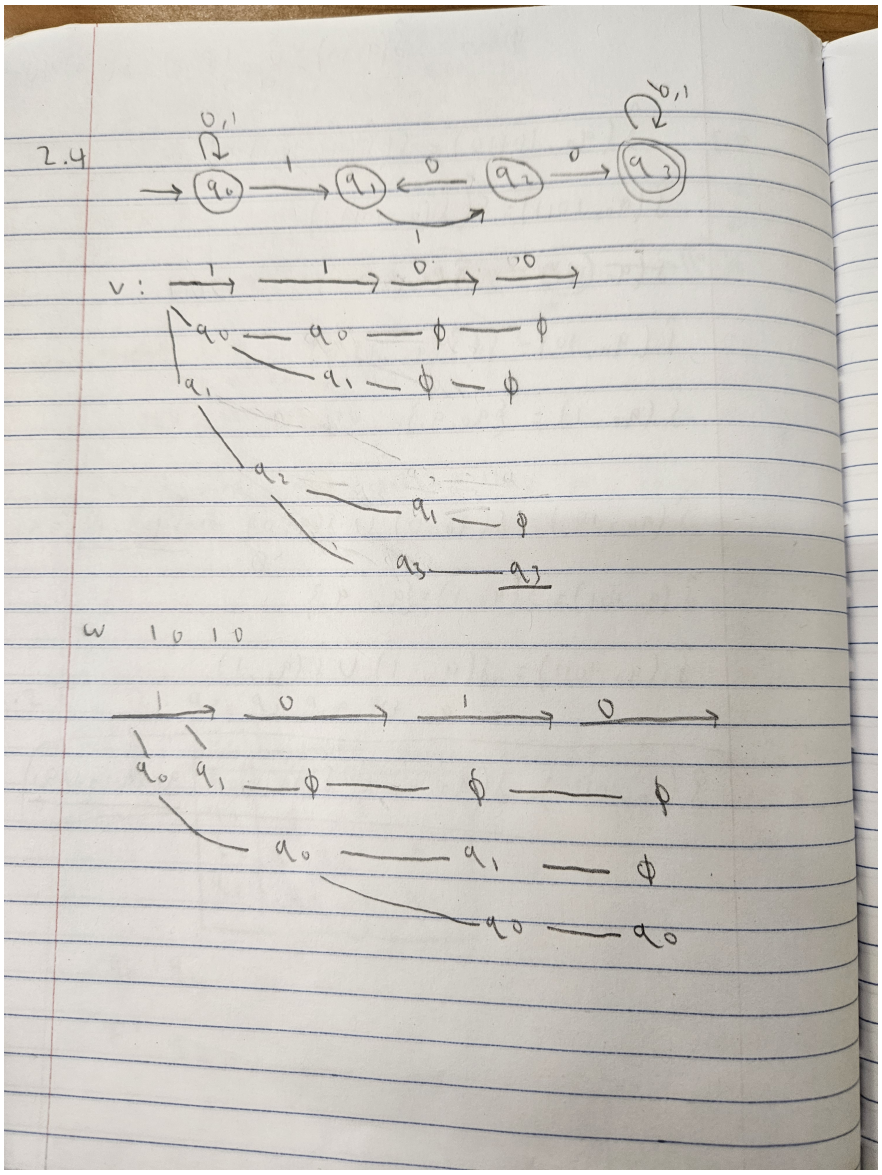
δ:

	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

$q_0: q_0$

$F: q_3$

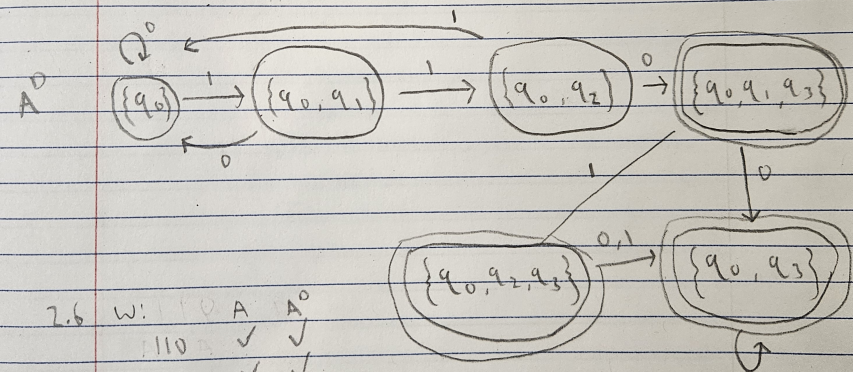

given $\delta(q, xa) = \bigcup_{i=1}^{n} \delta(p_i, a) = \delta(p_1, a) \cup \delta(p_2, a) \ldots$
2.3
$\hat{\delta}(q_0, 10110) = \delta(\hat{\delta}(q_0, 1011), 0)$
$\hat{\delta}(q_0, 1011) = \delta(\hat{\delta}(q_0, 101), 1)$
$\hat{\delta}(q_0, 101) = \delta(\hat{\delta}(q_0, 10), 1)$
$\hat{\delta}(q_0,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


2.4
0,1
1
0
0
0,1
1
v:
1
1
0
0
q_0 — q_0 — ϕ — ϕ
q_1 — ϕ — ϕ
q_2 — q_1 — ϕ
q_3 — q_3
w 1 0 1 0
1
0
1
0
q_0 q_1 — ϕ — ϕ — ϕ
q_0 — q_1 — ϕ
q_0 — q_0

2.5

A^D	1	0
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$* \{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0\}$
$* \{q_0, q_2\}$	$\{q_0, \emptyset\}$	$\{q_0, q_1, q_3\}$
$* \{q_0, q_1, q_3\}$	$\{q_0, q_2, q_3\}$	$\{q_0, \emptyset, q_3\} = \{q_0, q_3\}$
$* \{q_0, q_2, q_3\}$	$\{q_0, \emptyset, q_3\}$	$\{q_0, q_3, q_3\} = \{q_0, q_3\}$
$* \{q_0, \emptyset, q_3\}$	$\{q_0, q_3\}$	$\{q_0, q_3\}$

A^D



2.6

w:	A	A^D
110	✓	✓
1100	✓	✓
1010	X	X

I think there could be a smaller DFA accepting the same language but rejected words would not be treated the same. Accepted words shouldnt be effected if q_0 for instance wasnt in so many states