CPSC-406 Report

Jaime Song
Chapman University

02/16/25

Abstract

Contents

1 Introduction **1**

2 Week by Week **1**
2.1 Week 1 . . . 1
2.2 Week 2 . . . 2
2.3 Week 3 . . . 3
2.4 Week 4 . . . 8
2.5 Week 5 . . . 9
2.6 Week 6 . . . 11

3 Synthesis **13**

4 Evidence of Participation **14**

5 Conclusion **14**

1 Introduction

2 Week by Week

2.1 Week 1

Lab 1 Introduction to Automata Theory
Homework Question:
Characterize all the accepted words (i.e., describe exactly those words that get recognized).
Answer:

```
A word is accepted if it consists of the symbols 5 and 10, and the sum of all symbols is exactly
    25.
A word is accepted if it satisifes the equation:
5a + 10b = 25
a = num of times 5 appears
b = num of times 10 appears
a,b = non-negative integers
```

```
valid values of (a,b) are:
(5,0) = 5,5,5,5,5
(3,1) = 5,5,5,10
(1,2) = 5,10,10
```

Homework Question:
Characterize all the accepted words. Can you describe them via a regular expression?
Answer:

```
A word is accepted if it consists of one or more "pay" actions followed by a "push" and this
    sequence can repeat
any number of times
Expression:
(pay + push)+
```

2.2 Week 2

Lab 1 Deterministic and Non-deterministic Finite Automata (DFAs and NFAs)
Homework Question:
Determine for the following words, if they are contained in L1, L2, or L3
Answer:

```
w1 = 10011
  L1 = yes
  L2 = no
  L3 = no
w2 = 100
  L1 = no
  L2 = no
  L3 = no
w3 = 10100100
  L1 = yes
  L2 = yes
  L3 = yes
w4 = 1010011100
  L1 = yes
  L2 = no
  L3 = no
w5 = 11110000
  L1 = no
  L2 = yes
  L3 = yes
```

Homework Question:
Consider the paths corresponding to the words w1 = 0010, w2 = 1101, and w3 = 1100. For which of these words does their run end in the accepting state?
Answer:

```
Accepted: w1 = 0010, w2 = 1101
Not Accepted: w3 = 1100
```

ITALC Chapter 2.1 Summary

This section introduces finite automata through a real-world example: electronic money. The challenge is ensuring that digital currency can't be forged, duplicated, or spent multiple times. A bank plays a crucial role by issuing encrypted money files and keeping track of all valid transactions. While cryptography secures the money itself, protocols must prevent fraud and ensure transactions follow the intended process. The system involves three participants: the customer, the store, and the bank. The customer can either pay the store or cancel the transaction, returning the money to their account. The store can ship goods or redeem the money by sending it to the bank, which then transfers ownership. However, issues arise if the customer attempts to both pay and cancel, or if the store ships goods before confirming payment. To analyze these interactions, each participant's behavior is modeled as a finite automaton, where states represent different stages of a transaction and transitions correspond to actions taken. When combining these automata, the overall system can be examined for vulnerabilities. This method reveals flaws in poorly designed protocols, such as scenarios where the store ships goods but never gets paid. By using automata, we can validate protocols and ensure secure digital transactions.

2.3 Week 3

Lab 2
Programming (with) Automata
Exercise 2

---

dfa.py:

```
# a class for DFAs
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
    def run(self, w) :
        current_state = self.q0 # start at initial state
        for symbol in w: # process each character in the input word
            if (current_state, symbol) in self.delta:
                current_state = self.delta[(current_state, symbol)] # Move to next state
            else:
                return False # invalid transition, reject the word
        return current_state in self.F # accept if final state is in F
```

dfa_ex01.py:

```
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 = {0,1}
    Sigma1 = {'a', 'b'}
    delta1 = {(0, 'a'): 1, (0, 'b'): 0, (1, 'a'): 1, (1, 'b'): 0}
    q01 = 0
    F1 = {1}
    A1 = dfa.DFA(Q1, Sigma1, delta1, q01, F1)

    Q2 = {0, 1}
    Sigma2 = {'a', 'b'}
    delta2 = {(0, 'a'): 0, (0, 'b'): 1, (1, 'a'): 1, (1, 'b'): 1}
    q02 = 0
    F2 = {1}
    A2 = dfa.DFA(Q2, Sigma2, delta2, q02, F2)

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

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

__main__()
```

```
output:
DFA({0, 1}, {'b', 'a'}, {(0, 'a'): 1, (0, 'b'): 0, (1, 'a'): 1, (1, 'b'): 0}, 0, {1})
aaa: True
aab: False
aba: True
abb: False
baa: True
bab: False
bba: True
bbb: False


DFA({0, 1}, {'b', 'a'}, {(0, 'a'): 0, (0, 'b'): 1, (1, 'a'): 1, (1, 'b'): 1}, 0, {1})
aaa: False
aab: True
aba: True
abb: True
baa: True
bab: True
bba: True
bbb: True
```

Exercise 4

---

1. Accepting State: 4

```
The DFA accepts any string that contains at least one 'b' since reaching state 4 via any 'b' will
    keep it in an accepting state
```

2.

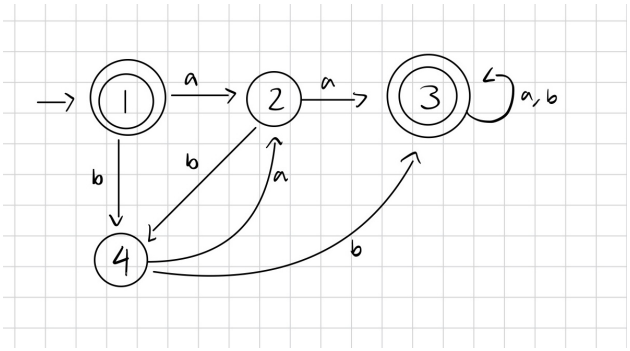

Figure 1:

---

dfa.py:

```
# a class for DFAs
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
    def run(self, w) :
        current_state = self.q0 # start at initial state
        for symbol in w: # process each character in the input word
            if (current_state, symbol) in self.delta:
                current_state = self.delta[(current_state, symbol)] # Move to next state
            else:
                return False # invalid transition, reject the word
        return current_state in self.F # accept if final state is in F

    def refuse(self):
        new_final_states = self.Q - self.F # complement final states
        return DFA(self.Q, self.Sigma, self.delta, self.q0, new_final_states)
```

dfa_ex03.py:

```
import dfa
def __main__() :
    Q = {1, 2, 3, 4} # set of states
    Sigma = {'a', 'b'} # alphabet
```

```
    delta = {
        (1, 'a'): 2, (1, 'b'): 4,
        (2, 'a'): 3, (2, 'b'): 4,
        (3, 'a'): 3, (3, 'b'): 3,
        (4, 'a'): 4, (4, 'b'): 4
    }
    q0 = 1 # initial state
    F = {4} # accepting states

    A = dfa.DFA(Q, Sigma, delta, q0, F) # instantiate DFA A

    A0 = A.refuse()
    test_cases = ["", "a", "aa", "aaa", "b", "ab", "ba", "abb", "aab"]

    print("Testing A (accepts strings with at least one 'b'):")
    for test in test_cases:
        print(f"String '{test}': {'Accepted' if A.run(test) else 'Rejected'}")

    print("\nTesting A0 (accepts strings without 'b'):")
    for test in test_cases:
        print(f"String '{test}': {'Accepted' if A0.run(test) else 'Rejected'}")

__main__()

output:
Testing A (accepts strings with at least one 'b'):
String '': Rejected
String 'a': Rejected
String 'aa': Rejected
String 'aaa': Rejected
String 'b': Accepted
String 'ab': Accepted
String 'ba': Accepted
String 'abb': Accepted
String 'aab': Rejected

Testing A0 (accepts strings without 'b'):
String '': Accepted
String 'a': Accepted
String 'aa': Accepted
String 'aaa': Accepted
String 'b': Rejected
String 'ab': Rejected
String 'ba': Rejected
String 'abb': Rejected
String 'aab': Accepted
```

ITALC: Exercise 2.2.4

```
a. Set of strings ending in 00
States:
* q0: Start state, accepts anything so far
* q1: Last character was 0
* q2: Last two characters were 00 (accepting state)
```

```
Transitions:
* From q0:
0 -> q1, 1-> q0
* From q1:
0 -> q2, 1-> q0
* From q2:
0 -> q2, 1-> q0

Accepting state: q2

b. Set of strings with three consecutive 0's
States:
* q0: Start state, no 0's seen yet
* q1: One 0 seen
* q2: Two consecutive 0's seen
* q3: Three consecutive 0's seen

Transitions:
* From q0:
0 -> q1, 1-> q0
* From q1:
0 -> q2, 1-> q0
* From q2:
0 -> q3, 1-> q0
* From q3:
0 -> q3, 1-> q3

Accepting state: q3

c. Set of strings with 011 as a substring
States:
* q0: Start state, no part of 011 seen yet
* q1: 0 seen
* q2: 01 seen
* q3: 011 seen (accepting state)

Transitions:
* From q0:
0 -> q1, 1-> q0
* From q1:
0 -> q1, 1-> q2
* From q2:
0 -> q1, 1-> q3
* From q3:
0 -> q3, 1-> q3

Accepting state: q3
```

Question:

```
How does a DFA remember whether it has seen the substring '01' during string processing, and how
    do the different states (q_0, q_1, q_2) represent the conditions required for acceptance?
```

2.4 Week 4

Homework 3
Operations on Automata
HW 1 (Extended Transition Function):

```
1. The automataon A^2 accepts all strings that contain an odd number of As. If a string has an
    even number of As (including 0) it is rejected

2. A^1:
(1,a) -> 2
(1,b) -> 4
(1,a) -> 4
(1,a) -> 4

(1,abaa) = 4

A^2:
(1,a) -> 2
(2,b) -> 1
(1,b) -> 3
(3,a) -> 3

(1,abba) = 3 (not final state so not accepted)
```

HW 2 (Product Automata:)
1.

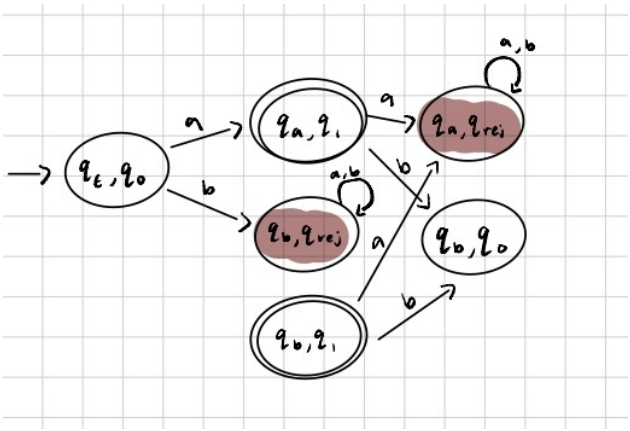

Figure 2:

```
\textbf{HW 2 (Product Automata:)}
\begin{lstlisting}
1.

2. A string is accpeted by A if and only if it is accepted by both A^1 and A^2
* By definition, A tracks the simultaneous behavior of A^1 and A^2
* A string w is accepted in A if and only if both automata reach a final state after processing w
* Since A is constructed with states (q^1, q^2), the final states of A are precisely those where
    both components are final
```

```
* This proves the equation

3. To change A to recognize L(A^2) U L(A^2) we can modify its final states
* Rather than requiring both automata to be in final state, we can accept if either automaton is
    in a final state
* change from:
F = F^2 x F^2
to:
F' = (F^1 x Q^2) U (Q^1 x F^2)

* means string is accepted by either A^1 or A^2
```

ITALC: Exercise 2.2.7

```
Proof:
Let A be a DFA, q a state of A such that \delta(q,a) = q for all a. We will prove by induction on
    the length of the input string w that d(q,w) = q for all w.

Base Case:
For w = \epsilon, we have d(q,\epsilon) = q by definition, so the base case holds.

Inductive Step:
Assume d(q,w) = q for some string w of length n. For w' = wa, we have:
d(q,w\prime) = \delta(d(q,w),a) = \delta(q,a) = q
Thus, d(q,w\prime) = q, and the inductive step holds.

By induction, d(q,w) = q for all w.
```

ITALC: Section 2.3

```
How does a nondeterministic finite automaton (NFA) process the string "00101" to accept it, and
    how does it differ from a deterministic finite automaton (DFA) in terms of state transitions?
```

2.5 Week 5

Homework 4
Determinization of NFAs
HW 1:

```
1. A DFA is a special case of an NFA. In an NFA, the transition function allows multiple possible
    states for a given input, while in the DFA, the transition function maps each state and input
    to a single state.
Since every DFA is inherently an NFA where the transition function always returns a unique next
    state, we can view A as an NFA without making any modifications. We reinterpret the DFA as an
    NFA where:
* The state set remains the same
* The transition function is modified to return singleton sets
* The start state and accepting state remain unchanged

2. Given a general DFA A = (Q, sigma, delta, q0, F) we can define an equivalent NFA A' = (Q',
    sigma', delta', q0', F') where:
Q' = Q
sigma remains the same
delta': Q' x sigma -> P(Q') is defined as:
```

```
delta'(q,a) = {delta(q,a)} for all q belongs to Q, a belongs to sigma
This ensures that each transition in A is preserved but is now viewed as returning a set of states
    rather than a single state
q'0 = q0 (initial state remains unchanged)
F' = F (final states remain unchanged)
```

HW 2:

```
1. Looking at the NFA:
* It starts at q0, where it loops on both 0 and 1
* A transition to q1, happens on input 1
* From q1, it can go to q2 on 0 or back to itself on 1
* From q2, it transitions to q3 on 0
* q3 is a final state and loops on both 0 and 1
The NFA recognizes the language of binary strings that contain 100 as a substring. The accepting
    state q3 is reached whenever the sequence 100 appears, and it stays in q3 accepting all
    further inputs

2. NFA A given by:
* Q = {q0, q1, q2, a3}
* Sigma = {0,1}
* Transition function delta:
delta(q0, 0) = {q0}
delta(q0, 1) = {q0,q1}
delta(q1, 0) = {q2}
delta(q1, 1) = {q1}
delta(q2, 0) = {q3}
delta(q2, 1) = {q1}
delta(q3, 0) = {q3}
delta(q3, 1) = {q3}
* Start state: q0
* Accepting states: F = {q3}

3. Step by step
  1. delta^(q0,1) = {q0,q1} (since q0 transitions to both itself and q1 on 1)
  2. delta^({q0, q1}, 0) = {q0, q2} (since q0 loops on 0 and q1 moves to q2)
  3. delta^({q0, q2}, 1) = {q0, q1} (since q0 loops on 1 and q2 transitions to q1 on 1)
  4. delta^({q0, q1}, 1) = {q0, q1} (same reason as step 1)
  5. delta^({q0, q1}, 0) = {q0, q2} (same reason as step 2)
Final states: {q0,q2} which does not include q3, so 101110 is rejected
```

---

4 and 5.

```
6. {q0,q2,q3} and {q0,q1,q3} can be combined into one state
```

ITALC: Section 3.1, 3.2.1, 3.2.2

```
Question: What does the Kleene star operation represent in the context of regular expressions?
```

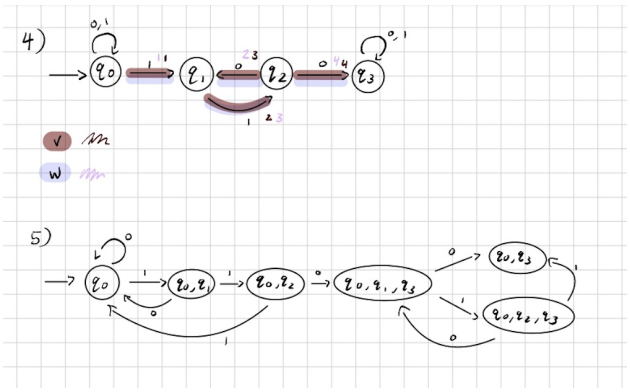

Figure 3:

2.6 Week 6

Homework 5 ITALC 3.2
Section 3.2.1
a. Here, $R_{ij}^{(0)}$ represents the regular expression for transitioning from state q_i to state q_j using exactly zero states as intermediates. These are simply based on the direct transitions:

$$
\begin{aligned}
R_{11}^{(0)} &= \emptyset \quad \text{(No direct loop on } q_1\text{)} \\
R_{12}^{(0)} &= 0 \quad \text{(From } q_1 \text{ to } q_2 \text{ on input 0)} \\
R_{13}^{(0)} &= \emptyset \quad \text{(No direct transition from } q_1 \text{ to } q_3\text{)} \\
R_{21}^{(0)} &= 1 \quad \text{(From } q_2 \text{ to } q_1 \text{ on input 1)} \\
R_{22}^{(0)} &= \emptyset \quad \text{(No direct loop on } q_2\text{)} \\
R_{23}^{(0)} &= 0 \quad \text{(From } q_2 \text{ to } q_3 \text{ on input 0)} \\
R_{31}^{(0)} &= \emptyset \quad \text{(No direct transition from } q_3 \text{ to } q_1\text{)} \\
R_{32}^{(0)} &= 1 \quad \text{(From } q_3 \text{ to } q_2 \text{ on input 1)} \\
R_{33}^{(0)} &= 0 \quad \text{(From } q_3 \text{ to itself on input 0)}
\end{aligned}
$$

b. Now we calculate the regular expressions using one intermediate state (state q_1).

$$
\begin{aligned}
R_{11}^{(1)} &= R_{11}^{(0)} \cup (R_{12}^{(0)} R_{21}^{(0)}) = \emptyset \cup (0 \cdot 1) = 01 \\
R_{12}^{(1)} &= R_{12}^{(0)} \cup (R_{12}^{(0)} R_{22}^{(0)}) = 0 \cup (0 \cdot \emptyset) = 0 \\
R_{13}^{(1)} &= R_{13}^{(0)} \cup (R_{12}^{(0)} R_{23}^{(0)}) = \emptyset \cup (0 \cdot 0) = 00 \\
R_{21}^{(1)} &= R_{21}^{(0)} \cup (R_{22}^{(0)} R_{21}^{(0)}) = 1 \cup (\emptyset \cdot 1) = 1 \\
R_{22}^{(1)} &= R_{22}^{(0)} \cup (R_{22}^{(0)} R_{22}^{(0)}) = \emptyset \cup (\emptyset \cdot \emptyset) = \emptyset \\
R_{23}^{(1)} &= R_{23}^{(0)} \cup (R_{22}^{(0)} R_{23}^{(0)}) = 0 \cup (\emptyset \cdot 0) = 0 \\
R_{31}^{(1)} &= R_{31}^{(0)} \cup (R_{32}^{(0)} R_{21}^{(0)}) = \emptyset \cup (1 \cdot 1) = 11 \\
R_{32}^{(1)} &= R_{32}^{(0)} \cup (R_{32}^{(0)} R_{22}^{(0)}) = 1 \cup (1 \cdot \emptyset) = 1 \\
R_{33}^{(1)} &= R_{33}^{(0)} \cup (R_{32}^{(0)} R_{23}^{(0)}) = 0 \cup (1 \cdot 0) = 10
\end{aligned}
$$

c. Now we calculate the regular expressions using up to two intermediate states (q_1 and q_2).

$$
\begin{aligned}
R_{11}^{(2)} &= 01 \cup (0 \cdot 0 \cdot 11) = 01 \cup 0011 \\
R_{12}^{(2)} &= 0 \cup (0 \cdot 0 \cdot 1) = 0 \cup 001 = 0 \\
R_{13}^{(2)} &= 00 \cup (0 \cdot 0 \cdot 0) = 00 \cup 000 = 00 \\
R_{21}^{(2)} &= 1 \cup (0 \cdot 1 \cdot 11) = 1 \cup 011 = 1 \\
R_{22}^{(2)} &= \emptyset \cup (0 \cdot 1 \cdot 1) = \emptyset \\
R_{23}^{(2)} &= 0 \cup (0 \cdot 1 \cdot 0) = 0 \\
R_{31}^{(2)} &= 11 \cup (1 \cdot 0 \cdot 11) = 11 \cup 1011 = 11 \\
R_{32}^{(2)} &= 1 \cup (1 \cdot 0 \cdot 1) = 1 \cup 101 = 1 \\
R_{33}^{(2)} &= 10 \cup (1 \cdot 0 \cdot 0) = 10 \cup 100 = 10
\end{aligned}
$$

d. The language of the automaton is defined by all possible strings leading to the final state q_3.

We observe: - q_3 is the only accepting state. - The simplest regular expression can be derived by considering all transitions leading to q_3. - The language consists of strings ending in 00 or 000, as they lead to the final state.

Thus, the regular expression for the language is:

$$(0^*1^*0^*0)$$

Section 3.2.1
c. Using up to two intermediate states (q_1 and q_2), we calculate the regular expressions for all state pairs:

$$
\begin{aligned}
R_{11}^{(2)} &= R_{11}^{(1)} \cup (R_{12}^{(1)} R_{21}^{(1)}) = \epsilon \cup (0 \cdot 1) = \epsilon \cup 01 \\
R_{12}^{(2)} &= R_{12}^{(1)} \cup (R_{12}^{(1)} R_{22}^{(1)}) = 0 \cup (0 \cdot \epsilon) = 0 \\
R_{13}^{(2)} &= R_{13}^{(1)} \cup (R_{12}^{(1)} R_{23}^{(1)}) = 1 \cup (0 \cdot 0) = 1 \cup 00 \\
R_{21}^{(2)} &= R_{21}^{(1)} \cup (R_{22}^{(1)} R_{21}^{(1)}) = 1 \cup (\epsilon \cdot 1) = 1 \cup 1 = 1 \\
R_{22}^{(2)} &= R_{22}^{(1)} \cup (R_{22}^{(1)} R_{22}^{(1)}) = \epsilon \cup (\epsilon \cdot \epsilon) = \epsilon \\
R_{23}^{(2)} &= R_{23}^{(1)} \cup (R_{22}^{(1)} R_{23}^{(1)}) = 0 \cup (\epsilon \cdot 0) = 0 \\
R_{31}^{(2)} &= R_{31}^{(1)} \cup (R_{32}^{(1)} R_{21}^{(1)}) = 1 \cup (1 \cdot 1) = 1 \cup 11 \\
R_{32}^{(2)} &= R_{32}^{(1)} \cup (R_{32}^{(1)} R_{22}^{(1)}) = 1 \cup (1 \cdot \epsilon) = 1 \\
R_{33}^{(2)} &= R_{33}^{(1)} \cup (R_{32}^{(1)} R_{23}^{(1)}) = \epsilon \cup (1 \cdot 0) = \epsilon \cup 10
\end{aligned}
$$

d. The language of the DFA consists of all possible strings that lead to the accepting state q_3. From the table, we observe that:

- State q_3 is the accepting state. - q_3 can be reached directly using 1 or through sequences like 00. - Using the repeating structure, the regular expression for the language can be written as:

$$1^*(00)^*1^*$$

This expression represents all valid sequences accepted by the DFA.

ITALC 4.4
Section 4.4.1

3 Synthesis

a.

	q_1	q_2	q_3
q_1			
q_2			
q_3			

1. **Mark accepting and non-accepting pairs**: - (q_1, q_3) and (q_2, q_3) since q_3 is accepting.

2. **Apply transitions**: - On input **0** and **1**, follow state transitions and check if they lead to distinguishable pairs. - Update the table accordingly.

3. **Final table**:

	q_1	q_2	q_3
q_1	ϵ	Indistinguishable	Distinguishable
q_2	Indistinguishable	ϵ	Distinguishable
q_3	Distinguishable	Distinguishable	ϵ

b. - **q_1 and q_2 are equivalent** because they are not distinguishable. - q_3 remains separate as it is distinguishable from all others.

Thus, the reduced DFA has: - **Two states**: $[q_1, q_2]$ and q_3. - **Transitions**: - $[q_1, q_2] \xrightarrow{0} [q_1, q_2]$ - $[q_1, q_2] \xrightarrow{1} q_3$ - $q_3 \xrightarrow{0} [q_1, q_2]$ - $q_3 \xrightarrow{1} [q_1, q_2]$ - **Start state**: $[q_1, q_2]$ - **Accepting state**: q_3

Section 4.4.2
a.

	A	B	C	D
A	ϵ			
B		ϵ		
C			ϵ	
D				ϵ

1. **Mark accepting and non-accepting pairs**: - If one state is accepting and the other is not, mark them as distinguishable.

2. **Apply transitions**: - Check where states transition on inputs **0** and **1**. - If transitions lead to distinguishable pairs, update the table.

3. **Final table after refinement**:

	A	B	C	D
A	ϵ	Indistinguishable	Distinguishable	Distinguishable
B	Indistinguishable	ϵ	Distinguishable	Indistinguishable
C	Distinguishable	Distinguishable	ϵ	Distinguishable
D	Distinguishable	Indistinguishable	Distinguishable	ϵ

b. - **C, F, I are equivalent** because they are all accepting states and indistinguishable. - **D and G are equivalent**, as they have the same transitions. - **E and H are equivalent**, as they behave identically.

Thus, the reduced DFA has: - **Five states**: $[A]$, $[B]$, $[C, F, I]$, $[D, G]$, $[E, H]$. - **Transitions**: - $[A] \xrightarrow{0} [B]$ - $[A] \xrightarrow{1} [E, H]$ - $[B] \xrightarrow{0} [C, F, I]$ - $[B] \xrightarrow{1} [E, H]$ - $[C, F, I] \xrightarrow{0} [D, G]$ - $[C, F, I] \xrightarrow{1} [H, E]$ - $[D, G] \xrightarrow{0} [E, H]$ - $[D, G] \xrightarrow{1} [C, F, I]$ - $[E, H] \xrightarrow{0} [A]$ - $[E, H] \xrightarrow{1} [C, F, I]$ - **Start state**: $[A]$ - **Accepting state**: $[C, F, I]$

ITALC: 3.2 and 4.4

```
Question: How does the minimization of a DFA impact its computational efficiency, and are there
    cases where minimizing a DFA could be detrimental rather than beneficial?
```

---

4 Evidence of Participation

5 Conclusion

References

[BLA] Author, Title, Publisher, Year.